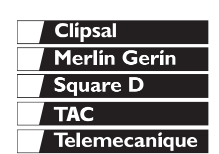
Unity Pro
语言和程序结构参考手册

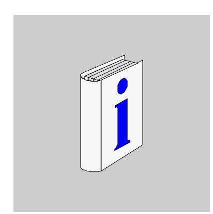




目录



关于本书



内容预览

文档范围

本手册描述了使用 Unity Pro V1.0 编程工作时，进行 Premium、 Atrium 和 Quantum PLC 编程所需的基础内容。

有效性声明

本手册中的数据和演示并不具有约束力。我们保留因进行产品持续开发而修改产品的权利。本文档中所包含的信息会在未给出事先通知的情况下进行更改，施耐德公司不应该对此承担责任。

与产品相关的警告

施耐德电气公司对本文档中所出现的错误不承担任何责任。如果您对本文有任何改进或更改建议，请通知我公司。

在未获得施耐德电气公司明确书面许可的情况下，本文档的任何部分都不能以任何电子或者机械形式被复制，包括影印。

在安装和使用本产品的过程中，必须遵守所有相关的州，地区，地方安全法规。为了保证安全，并确保文献系统资料的一致性，部件的维修只应由厂商来进行。

在带有技术安全要求的应用场合使用控制器时，请遵循相关说明。

如果不使用施耐德电气公司的软件，或者在我公司的硬件产品中未能使用经过核准的软件，可能会造成伤害，损伤，或者导致错误的操作结果。

如果不遵守与本产品相关的警告，可能会导致伤害，或者损坏设备。

用户建议

非常欢迎您对本文档提出相关建议。您也可以通过电子邮件 TECHCOMM@modicon.com 和我们联系。

Unity Pro 的一般介绍

I

介绍

本节内容

本节介绍了通过 Unity Pro 创建的应用程序的常规设计和行为。

本部分内容

本部分包含以下各章：

介绍

1

介绍

概述

本节介绍了通过 Unity Pro 创建的项目的常规设计和行为。

本章内容

本章包含以下内容：

Unity Pro 的特性

硬件平台

Unity Pro 支持如下硬件平台：
- Quantum
- Premium
- Atrium

编程语言

Unity Pro 为创建用户程序提供了如下编程语言：
- 功能块图 FBD
- 梯形图语言 LD
- 指令表 IL
- 结构化文本 ST
- 顺序控制 SFC

在同一个项目中，可以同时使用所有这些编程语言。
这些编程语言都符合 IEC 61131-3。

功能块库

在 Unity Pro 扩展功能块库中所包含的功能块，既有用于简单布尔运算的功能块，也有用于字符串和数组运算的功能块，另外还有用于控制复杂控制循环的功能块。为了便于用户把握各种功能块，不同的功能块被归纳到各个库中，并划分为不同的系列。
功能块可以用在编程语言 FBD，LD，IL 和 ST 中。

程序的元素

一个程序可以包含：
- 一个主任务 (MAST)
- 一个快速任务 (FAST)
- 一到四个辅助任务 (AUX)
- 多个分配了定义任务的代码段
- 用来处理时间控制事件的代码段 (Timerx)
- 用来处理硬件控制事件的代码段 (EVTx)
- 子程序代码段 (SR)

软件包

有以下可用的软件包：
- Unity Pro L
- Unity Pro XL
- Unity Studio
- Unity 开发版本 (UDE)

功能范围

下面的表格给出了个体软件包的主要特性：

	Unity Pro L	Unity Pro XL	Studio
编程语言			
功能块图 FBD	+	+	+
梯形图语言 LD	+	+	+
指令表 IL	+	+	+
结构化文本 ST	+	+	+
顺序语言 SFC	+	+	+
库			
标准库	+	+	+
控制库	-	+ (Premium：仅限于 57 5••) (Quantum：所有 CPUs)	+
控制库	+	+	+
诊断库	+	+	+
I/O 管理库	+	+	+
系统库	+	+	+
运动库	+	+	+
TCP Open 库	+	+	+
过时库	+	+	+
常规内容			
创建及使用数据结构和字段	+	+	+
创建及使用导出功能块 (DFB)	+	+	+
带有结构和 / 或功能视图的项目浏览器	+	+	+
管理访问权限	+	+	+
操作屏	+	+	+
诊断浏览器	+	+	+
系统诊断	+	+	+
项目诊断	+	+	+
旧数据转换器	+	+	+
管理多工作站	+	+	+
支持的平台	-	-	+

	Unity Pro L	**Unity Pro XL**	**Studio**
Premium	所有 CPU **除了** P57 554 P57 5634	所有 CPU	所有 CPU
Quantum	低端 CPU	所有 CPU	所有 CPU
Atrium	所有 CPU	所有 CPU	所有 CPU
模拟器	+	+	+
开放性			
超链接	+	+	+
VBA 脚本	+	+	+
VBA IDE	-	-	+
在软件包中所包含的软件组件			
XBTL1000	+	+	+
PowerSuite	+	+	+
OFS	-	-	+
其他	-	-	Unity Pro XL
Unity Studio 管理器	-	-	+
Visio2002 Enterprise	-	-	+
上下文帮助和 PDF 文档	+	+	+
EXEC 装载器 + HW 固件	+	+	+

图例说明：
+ = 可用
- = 不可用

命名规则

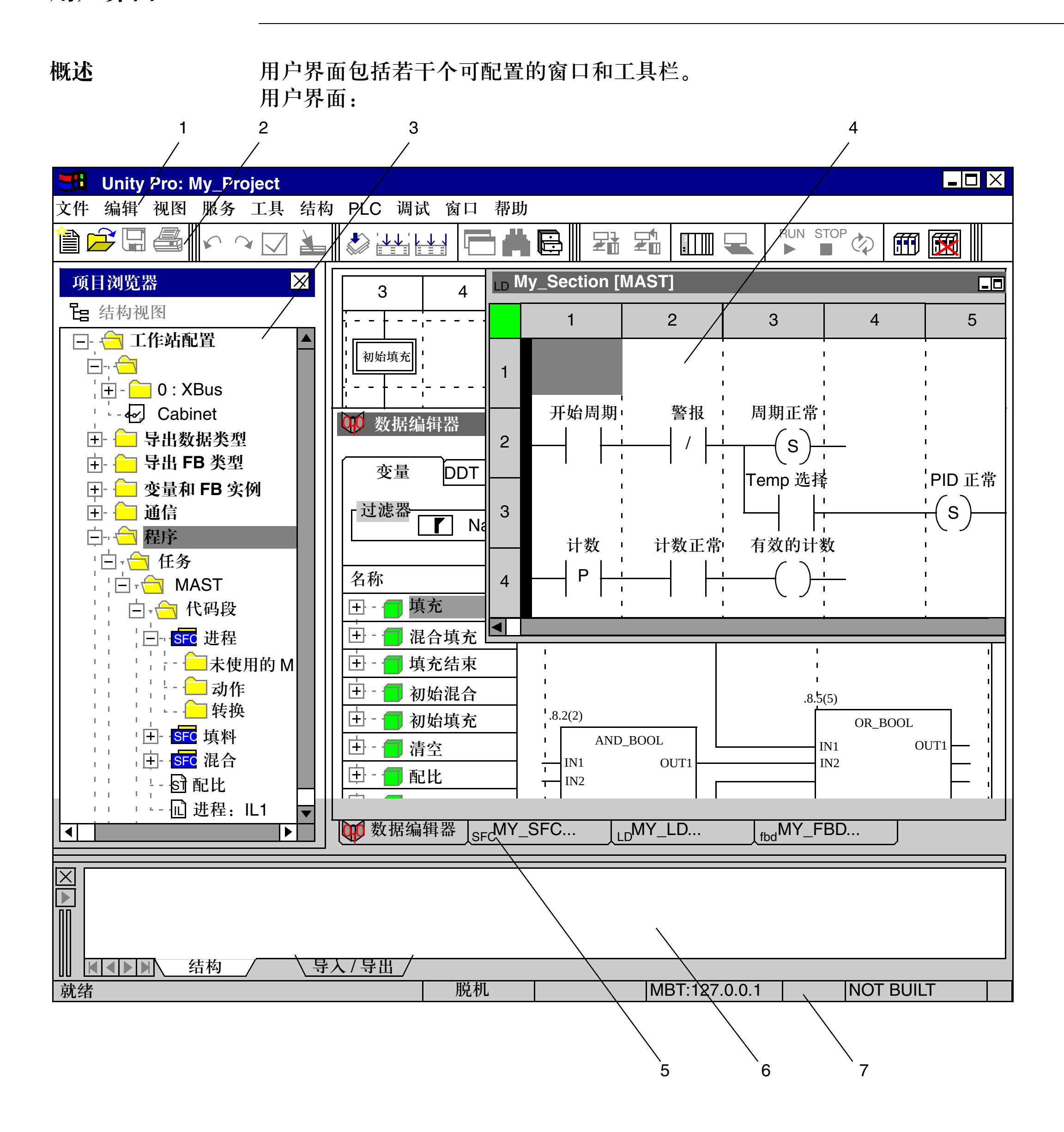
在后面的文档中，“Unity Pro L” 和 “Unity Pro XL”统称为 “Unity Pro”。

用户界面

概述

用户界面包括若干个可配置的窗口和工具栏。

用户界面：



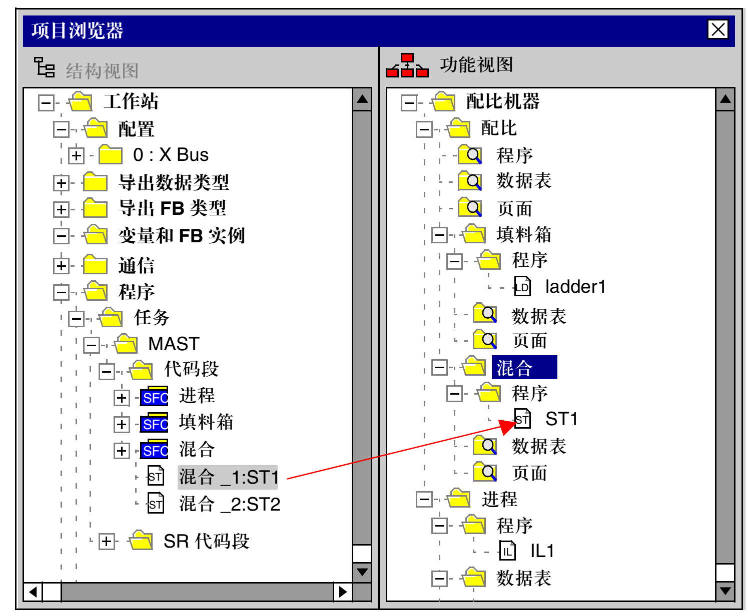
图例说明：

号码	介绍
1	菜单栏
2	工具栏
3	项目浏览器
4	编辑器窗口（编程语言编辑器，数据编辑器，等等）
5	用来直接访问编辑器窗口的选项卡
6	信息窗口（提供关于已发生错误，信号跟踪，导入功能等方面的信息）
7	状态栏

项目浏览器

介绍

项目浏览器显示所有项目参数。浏览器的视图可以通过结构 (拓扑) 和 / 或功能视图的形式显示出来。



结构试图：

在结构视图中，项目浏览器提供了如下功能：
- 创建和删除元素
- 代码段符号显示了代码段的编程语言，还可以显示它是否受保护
- 浏览元素属性
- 创建用户目录
- 启动不同的编辑器
- 开启导入 / 导出功能

功能试图：

在功能试图中，项目浏览器提供了如下功能：
- 创建功能组件
- 从结构视图中，通过拖放操作，插入代码段，动态数据表，操作员屏幕等内容。
- 创建代码段
- 浏览元素属性
- 启动不同的编辑器
- 代码段符号显示了编程语言和其他属性

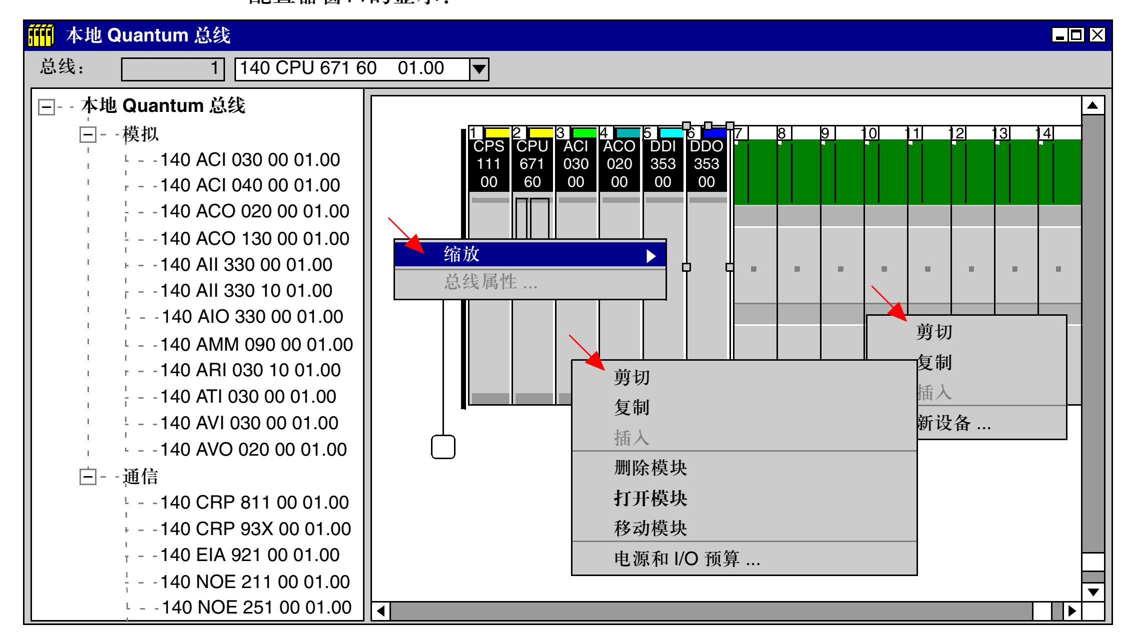
配置器

配置器窗口

配置器窗口内又分为两个窗口：

- 目录窗口
  可以在该窗口中选择一个模块，并通过拖放操作将其直接插入到 PLC 配置的图形演示中去。
- PLC 配置的图形显示

配置器窗口的显示：



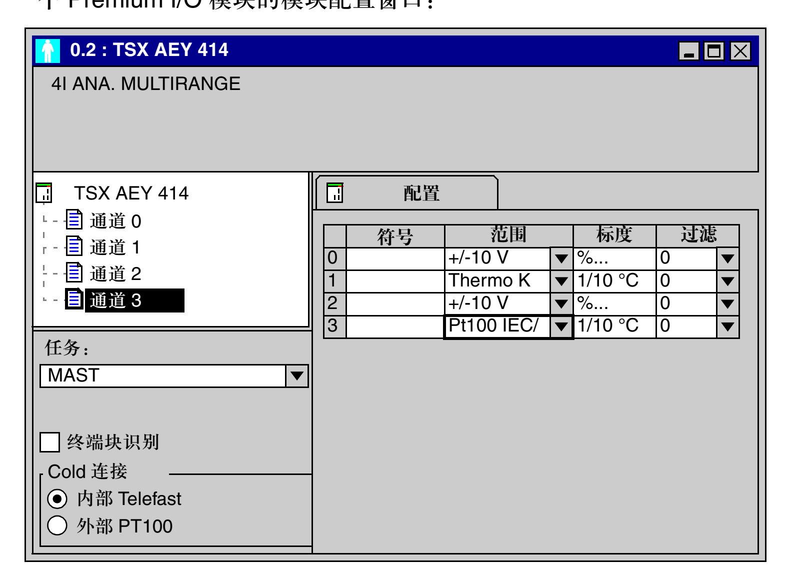
用户可以通过变换鼠标指针的位置来调用以下快捷菜单 (右键) 之一：

- 如果鼠标指针位于背景处：
  - 选择不同的缩放比例
- 如果鼠标指针位于模块上：
  - 实现编辑功能 (删除，复制，移动)
  - 打开用来定义模块特有参数的模块配置
  - 显示 I/O 属性和总电流
- 如果鼠标指针位于空插槽上：
  - 从目录中插入一个模块
  - 插入一个预先复制的模块，其中包括该模块预先定义的属性

模块配置

模块配置窗口 (通过模块快捷菜单或者双击模块来调用) 用来配置模块。它还包括通道选择，为所选通道选择功能，分配 State RAM 地址 (仅用于 Quantum) 等内容。

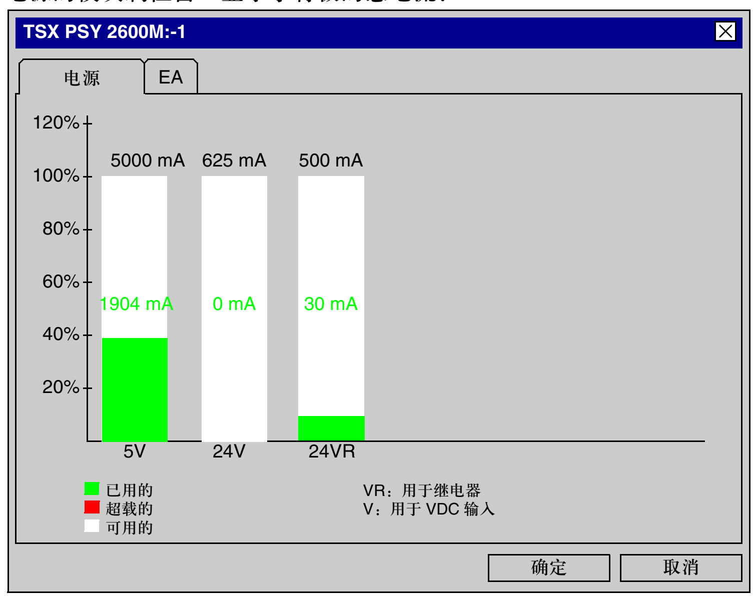
一个 Premium I/O 模块的模块配置窗口：



模块属性

模块属性窗口 (通过模块快捷菜单来调用) 显示了模块属性，比如功率消耗， I/O 点的数量 (仅限于 Premium)，等等。

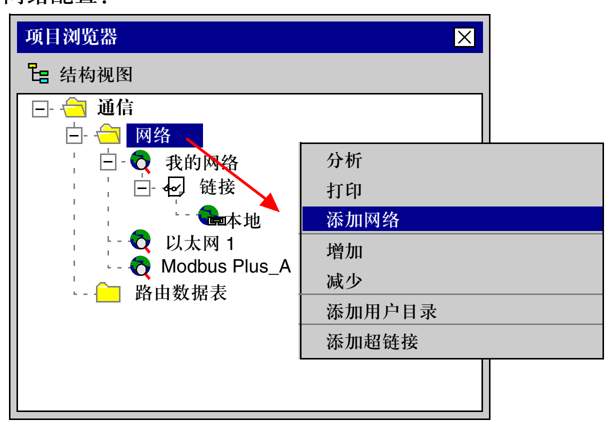
电源的模块属性窗口显示了背板的总电流：



网络配置

网络配置通过通信文件夹进行调用。

网络配置：



用户可以通过网络配置窗口：

- 创建网络
- 进行网络分析
- 打印输出网络配置

以下是一个配置网络的窗口：



配置完成以后，要为网络分配一个通信模块。

数据编辑器

介绍

数据编辑器具有以下功能：

- 对变量实例进行声明
- 对导出数据类型 (DDT) 进行定义
- 对基本和导出功能块 (EFB/DFB) 进行实例声明
- 对导出功能块 (DFB) 参数进行定义

在数据编辑器的所有选项卡中都可以使用以下功能：

- 复制，剪切，粘贴
- 扩展 / 缩进结构化数据
- 根据类型，符号，地址等进行分类
- 过滤器
- 插入，删除和更改栏位
- 在数据编辑器和程序编辑器之间进行拖放操作
- 取消上一次更改
- 导出 / 导入

变量

变量选项卡用来声明变量。
变量选项卡：

数据编辑器

变量 | DDT 类型 | 功能块 | DFB 类型

过滤 名称 * ☑ 基本的 ☐ 导出的 ☐ 导出的 I/O

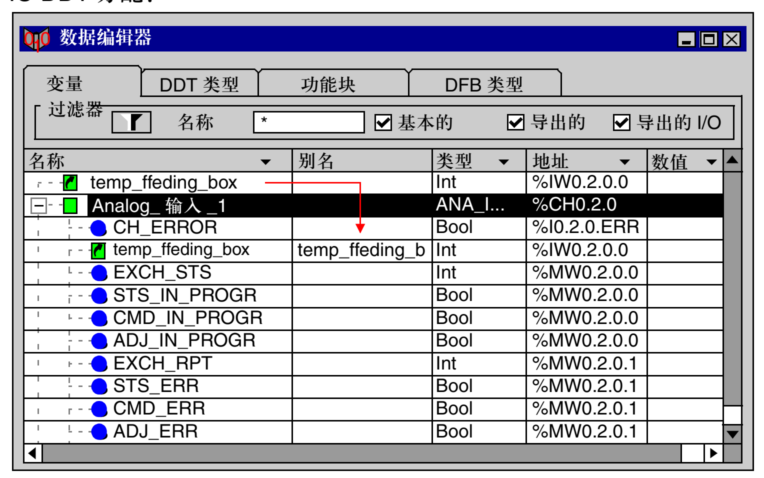
名称	类型	地址	数值	注释
工作站 1	Bool			
Filling_up	Bool			
Mixer_filling_u	Byte			
End_of_filling	Date			
Initial_mixer	Dint			
Initial_filling	Dt			
Empty_mixer	DWord			
Proportioning	EBoot			
Fill_up_A	Int			
Fill_up_B	Bool			
Sync_1	Dint			
Sync_2	Dint			
Fill_up_C	Bool			
Start_mixer	Bool			

有以下可用功能：
- 为变量定义符号
- 分配数据类型
- 为导出数据类型提供选择对话框
- 分配一个地址
- 自动用符号表示 I/O 变量
- 为一个初始值赋值
- 输入一个注释
- 在数据属性窗口中浏览一个变量的所有属性

取决于硬件的数据类型 (IO DDT)

IO DDT 用来把一个模块的完整 I/O 结构分配给一个个体变量。

IO DDT 分配：



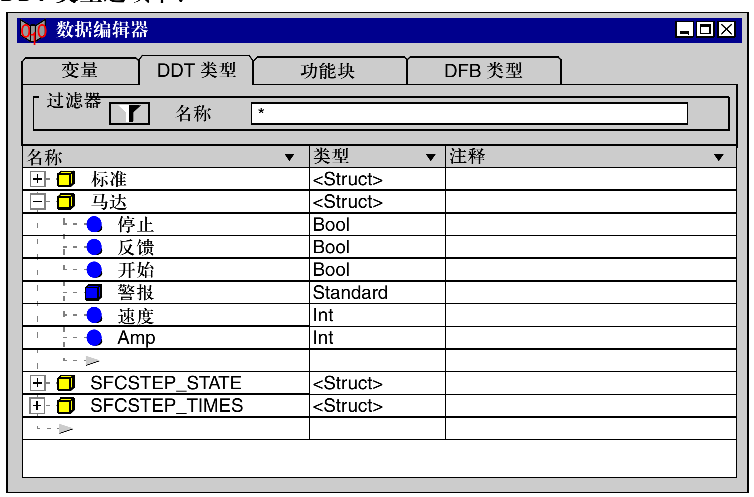
有以下可用功能：
- 完整的 I/O 结构可以通过 IO DDT 分配给个体变量。
- 在输入变量地址以后，系统会为结构的所有元素自动分配正确的输入 / 输出位或字。
- 因为地址分配的操作可以稍后进行，所以用户可以创建一个简单的标准模块，并在以后再定义名称。
- 用户可以为 I/O 结构的所有元素指定一个别名。

导出数据类型 (DDT)

DDT 类型选项卡用来定义导出数据类型 (DDT)。

导出数据类型是对任何来自已经定义的数据类型 (基本的或者导出的) 的结构或者数组的定义。

DDT 类型选项卡：



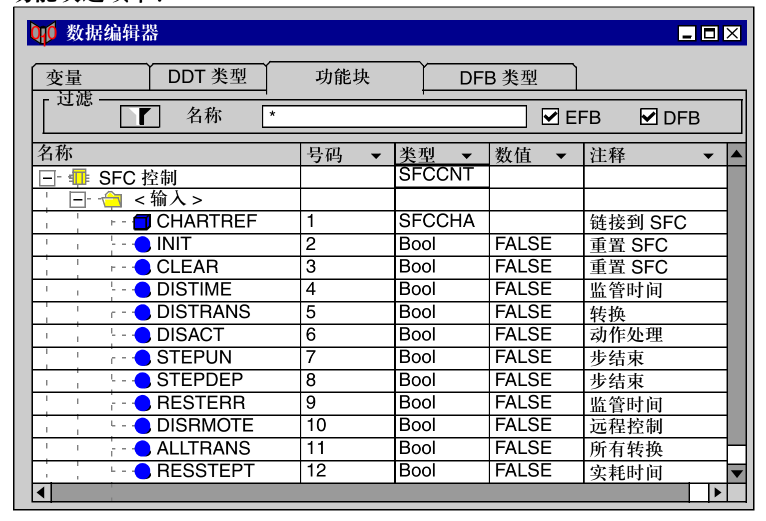
有以下可用功能：

- 嵌套 DDT 的定义 (最多 8 级)
- 数组的定义，最多有 6 维
- 初始值的赋值
- 地址分配
- 输入一个注释
- 导出数据类型分析
- 将导出数据类型分配给一个库
- 在数据属性窗口内浏览一个导出数据类型的所有属性

功能块

功能块选项卡用来对基本和导出功能块 (EFB/DFB) 的实例进行声明。

功能块选项卡：



有以下可用的功能：

- 显示在项目中所使用的功能块
- 为项目中所使用的功能块定义一个符号
- 自动激活在项目中所定义的符号
- 输入一个关于功能块的注释
- 浏览功能块的所有参数 (输入 / 输出)
- 将一个初始值赋给功能块输入 / 输出

DFB 类型

DFB 类型选项卡用来定义导出功能块 (DFB) 参数。

DFB 逻辑可以在 FBD，LD，IL 或者 ST 编程语言的一个或多个代码段内直接创建。

DFB 类型选项卡：



有以下可用的功能：

- 定义 DFB 名称
- 定义 DFB 的所有参数，比如：
  - 输入
  - 输出
  - VAR_IN_OUT (联合输入 / 输出)
  - 私有变量
  - 公共变量
- 为 DFB 参数分配数据类型
- 为导出数据类型提供选择对话框
- 为一个初始值赋值
- 嵌套 DFB
- 在一个 DFB 中使用若干个代码段
- 为 DFB 和 DFB 参数输入一个注释
- 对定义的 DFB 进行分析
- 版本管理
- 将定义的 DFB 分配给一个库

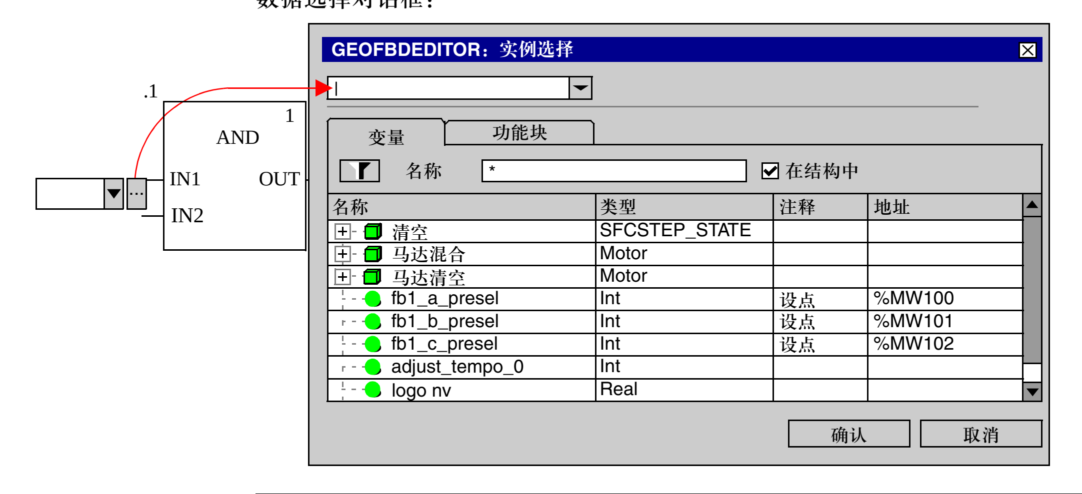
数据的使用

通过数据编辑器所创建的数据类型和实例可以插入到（取决于上下文）编程编辑器中去。

有以下可用的功能：

- 访问所有编程语言编辑器
- 只有兼容的数据会被显示出来
- 浏览根据所属库来排列的功能，功能块，过程和导出数据类型。
- 在编程过程中可以对实例进行声明

数据选择对话框：



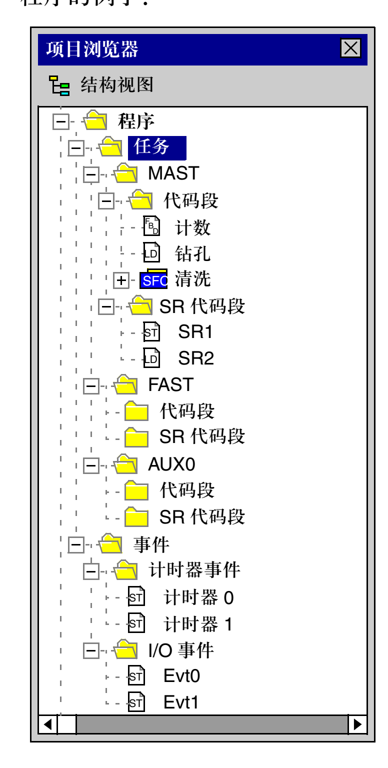
程序编辑器

介绍

可以从以下各项创建程序：

- **任务**，以周期或者循环方式执行。
  可以从以下各项创建任务：
  - 代码段
  - 子程序
- **事件处理**，要在所有其他任务之前执行。
  可以从以下各项创建事件处理：
  - 用来处理时间控制事件的代码段
  - 用来处理硬件控制事件的代码段

程序的例子：



任务

Unity Pro 支持多重任务 (多重任务处理)。

各个任务以 “并行”和独立的方式执行，其执行优先级被 PLC 所控制。用户可以根据不同的需要对任务进行调整，从而使其成为构建项目的有力工具。

可以从以下各项来构建多重任务项目：

- 一个主任务 (MAST)
  主任务以循环或者周期的方式执行。
  它形成程序的主代码段，并以顺序的方式执行。
- 一个快速任务 (FAST)
  快速任务以周期的方式执行。它的优先级比主任务高。快速任务用于以快速和周期方式执行的进程。
- 一到四个辅助任务 (AUX)
  辅助任务以周期的方式执行。它们用于慢速处理，优先级最低。

在构建项目的过程中，也可以只使用一个任务。在这种情况下，只有主任务是有效的。

事件处理

事件处理在事件代码段出现。事件代码段比所有其他任务代码段的优先级都高。它们适合用于那些在触发了一个事件之后需要极短反应时间的处理过程。

对于事件处理过程来说，有以下可用的代码段类型：

- 用来处理时间控制事件的代码段 (Timerx 代码段)
- 用来处理硬件控制事件的代码段 (Evtx 代码段)

它支持以下编程语言：

- FBD (功能块图)
- LD (梯形图语言)
- IL (指令表)
- ST (结构化文本)

代码段

代码段是自治的程序单元，项目逻辑是在代码段中创建的。代码段是按照项目浏览器 (结构试图) 中给出的顺序执行的。代码段与一个任务相关联。

同一个代码段不能同时隶属于一个以上的任务。

它支持以下编程语言：

- FBD (功能块图)
- LD (梯形图语言)
- SFC (顺序功能图)
- IL (指令表)
- ST (结构化文本)

子程序

子程序是在子程序代码段中作为单独的单元来创建的。
子程序是从代码段或者另外的子程序中被调用的。
嵌套的级别最多可以有 8 级。
子程序不能调用自己 (非递归)。
子程序会分配给一个任务。同一个子程序不能被不同的任务所调用。
它支持以下编程语言：
- FBD (功能块图)
- LD (梯形图语言)
- IL (指令表)
- ST (结构化文本)

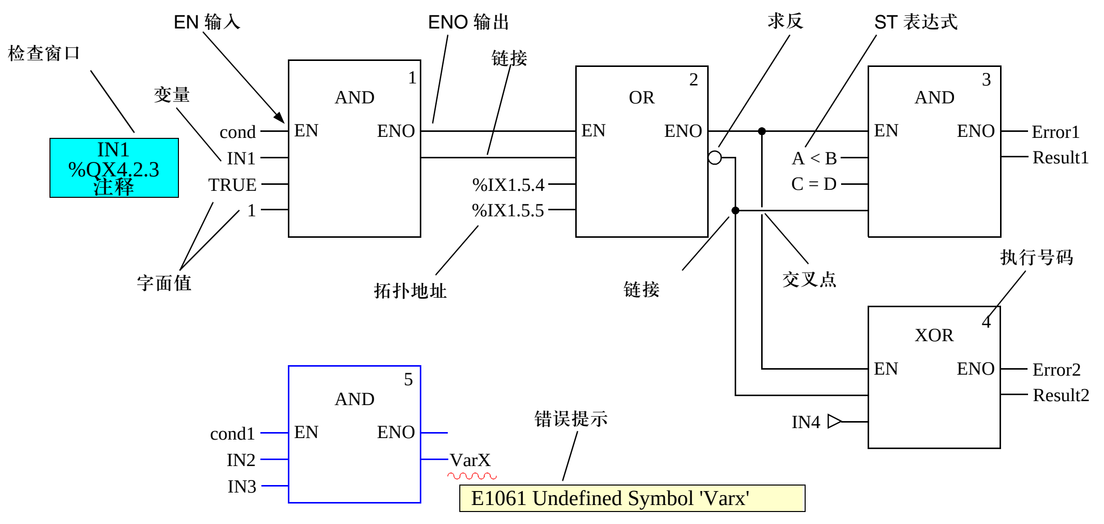
功能块图 FBD

介绍

FBD 编辑器用于符合 IEC 61131-3 的图形功能块编程。

演示

FBD 代码段的演示：



对象

用户可以借助 FBD (功能块图) 编程语言的对象把一个程序段划分为若干个：

- 基本功能 (EF)
- 基本功能块 (EFB)
- 导出功能块 (DFB)
- 功能程序
- 子程序
- 跳转
- 链接
- 实际参数
- 文本对象，给出逻辑注释

属性

FBD 代码段带有一个背景网格。一个网格单元由十个网格点组成。网格单元是 FBD 代码段内两个对象之间最小的距离。 FBD 编程语言不是面向单元的，但是对象可以通过网格坐标来进行对齐。

一个 FBD 代码段包含 300 个水平网格点 (=30 个网格单元) 以及 230 个垂直网格点 (=23 个网格单元)。

可以通过鼠标或者键盘输入程序代码。

输入帮助

FBD 编辑器提供了以下输入帮助：

- 用于对所需对象进行快速便捷访问的工具栏
- 在创建程序的时候，进行直接的语法和语义检查
  - 错误的功能和功能块会用蓝色显示出来
  - 未知字 (比如未经声明的变量) 或者不匹配的数据类型会用红色的波浪线标出
  - 在错误提示中给出简短的错误说明
- 表格显示的 FFB
- 实际的参数可以以符号或者拓扑地址的形式输入和显示
- 不同的缩放比例
- 链接跟踪
- 链接线路的优化
- 检查窗口的显示

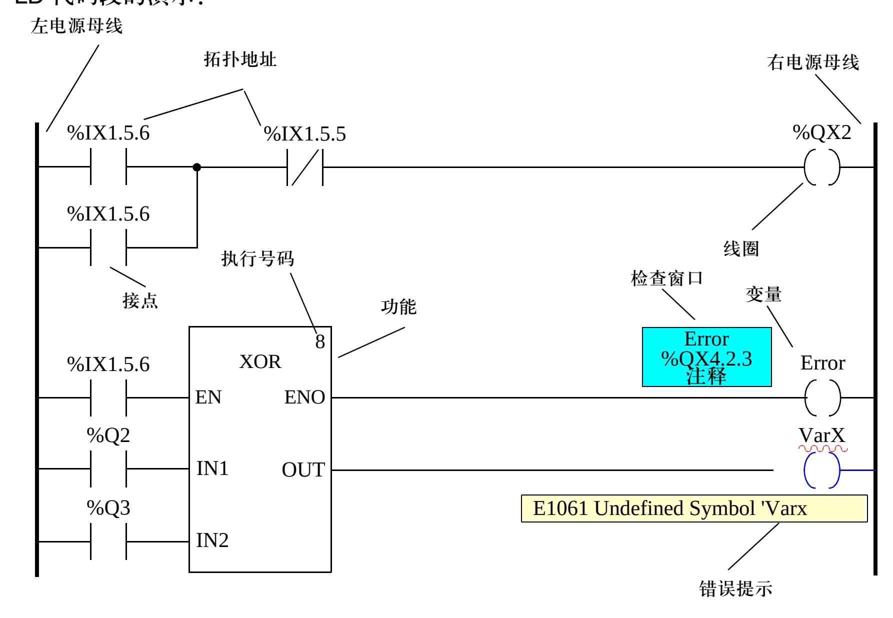
梯形图 (LD) 语言

介绍

LD 编辑器用于符合 IEC 61131-3 的梯形图编程。

演示

LD 代码段的演示：



对象

LD 编程语言对象帮助用户将一个代码段划分为若干个：
- 触点
- 线圈
- 基本功能 (EF)
- 基本功能块 (EFB)
- 导出功能块 (DFB)
- 功能程序
- 控制元素
- 对代表某一个 IEC 61131-3 扩展内容的功能块进行操作和比较
- 子程序调用
- 跳转
- 链接
- 实际的参数
- 文本对象，给出逻辑注释

属性

LD 代码段带有一个背景网格，该网格将代码段划分为行和列。 LD 编程语言是面向网格单元的，也就是说，在每一个网格单元中只能放置一个对象。
LD 代码段中可以定义 11-64 列和 17-2000 行。
可以通过鼠标或者键盘输入程序代码。

输入帮助

LD 编辑器提供了以下输入帮助：
- 可以从工具栏，菜单中选择对象，也可以借助快捷键直接进行选择
- 在创建程序的时候，直接进行语法和语义检查
  - 错误的对象会用蓝色显示出来
  - 未知字 (比如未经声明的变量) 或者不匹配的数据类型会用红色的波浪线标出
  - 在错误提示中给出简短的错误说明
- 表格显示的 FFB
- 实际的参数可以以符号或者拓扑地址的形式被输入和显示
- 不同的缩放比例
- FFB 链接的跟踪
- 优化 FFB 的链接线路
- 检查窗口的显示

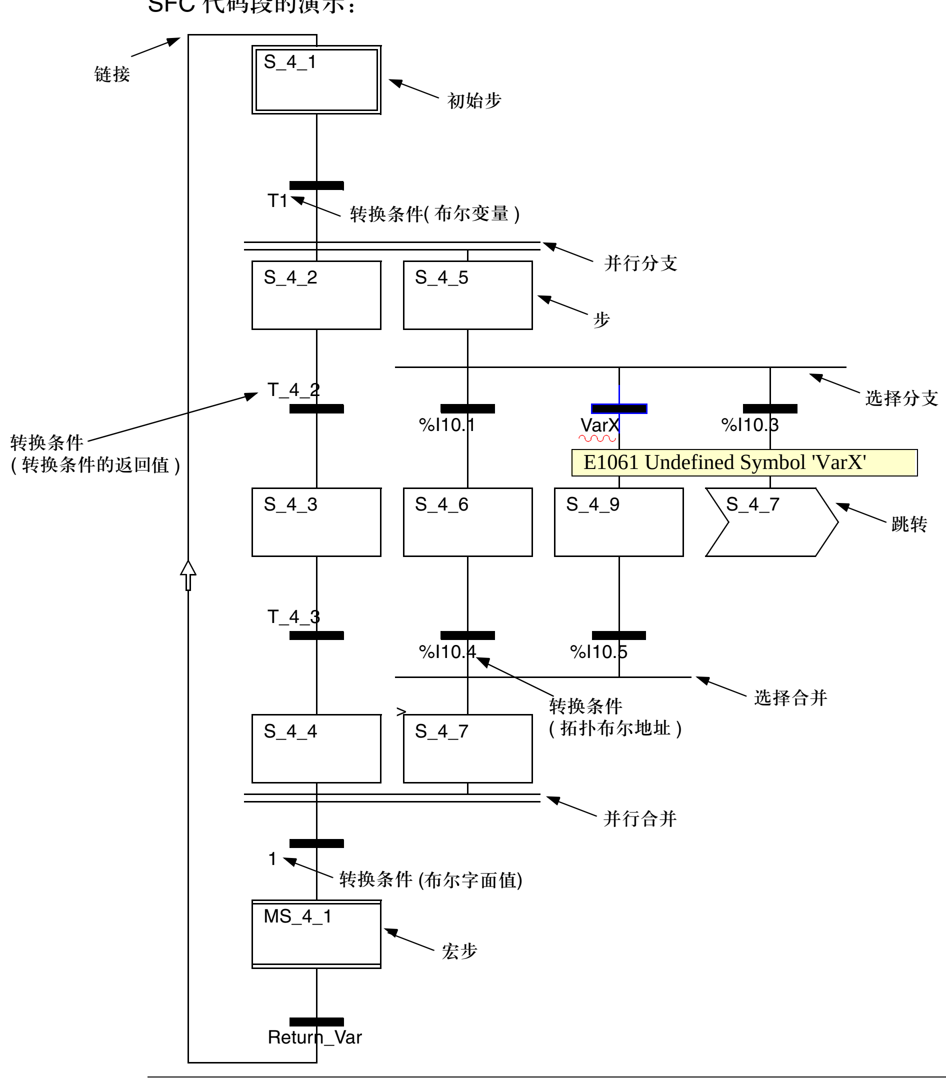
关于顺序功能图 (SFC) 的常规信息

介绍

该代码段描述了符合 IEC 61131-3 的顺序功能图 (SFC)。
通过显示使能步骤，可以消除 IEC 的限制，进而使用户能够使用诸如多令牌，多重初始步，与并行串之间进行跳转等功能。

演示

SFC 代码段的演示：



对象

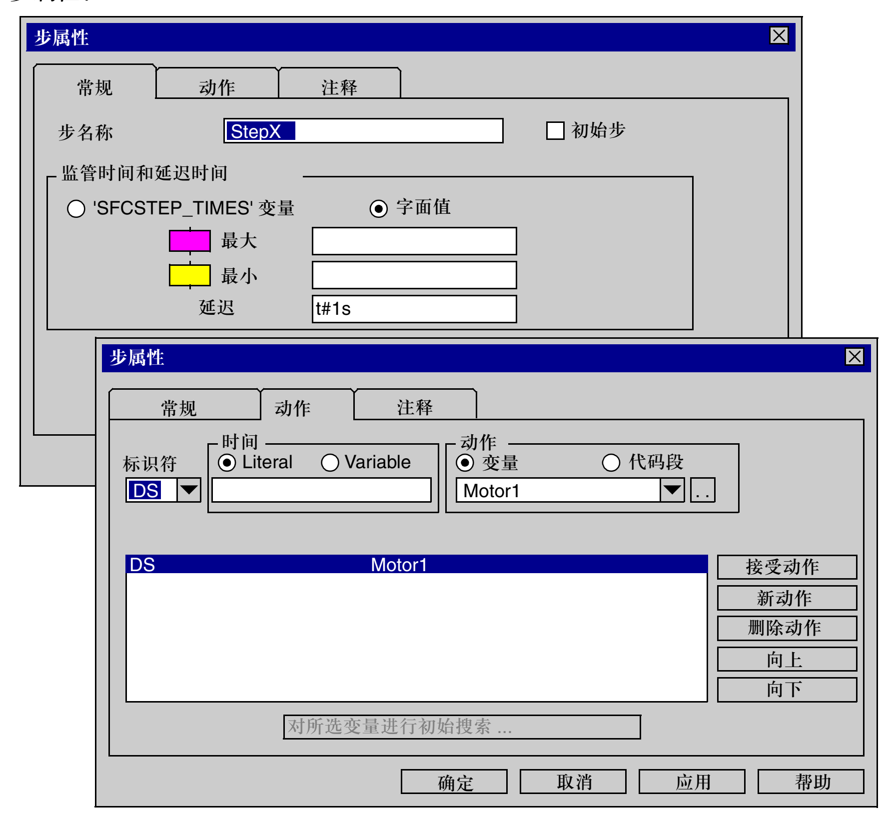
SFC 代码段为生成程序提供了如下对象：
- 步
- 宏步 (嵌入了子步的顺序)
- 转换 (转换条件)
- 转换代码段
- 动作代码段
- 跳转
- 链接
- 选择顺序
- 并行顺序
- 文本对象，给出逻辑注释

输入帮助

SFC 编辑器提供了以下输入帮助：
- 用来对所需对象进行快速便捷访问的工具栏
- 自动步编号
- 直接访问动作和转换条件
- 在创建程序的时候，直接进行语法和语义检查
  - 错误的对象会用蓝色显示出来
  - 未知字 (比如未经声明的变量) 或者不匹配的数据类型会用红色的波浪线标出
  - 在错误的对象会用蓝色显示出来
- 不同的缩放比例
- 显示 / 隐藏分配的动作
- 链接跟踪
- 链接线路的优化

步属性

步属性：



步属性通过一个对话框进行定义，该对话框具有以下功能：
- 定义初始步
- 定义诊断时间
- 步注释
- 分配动作及其标识符

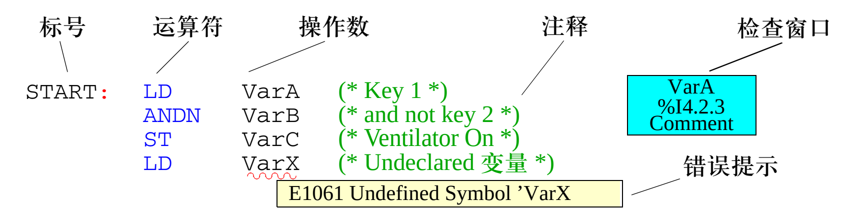
指令表 IL

介绍

IL 编辑器用于符合 IEC 61131-3 的指令表编程。

演示

IL 代码段演示：



对象

指令表包含一系列指令。每一个指令都会在新的一行开始，并含有以下内容：
- 一个运算符
- 如果需要的话，会有一个限定词
- 如果需要的话，会有一个或多个操作数
- 如果需要的话，会有一个作为跳转目标标记
- 如果需要的话，会有一个关于逻辑的注释

输入帮助

IL 编辑器提供了以下输入帮助：
- 在创建程序的时候，直接进行语法和语义检查
  - 用颜色显示出关键字和注释
  - 未知字 (比如未经声明的变量) 或者不匹配的数据类型会用红色的波浪线标出
  - 在快速信息 (错误提示) 中给出简短错误说明
- 表格显示功能和功能块
- 功能和功能块的输入辅助功能
- 操作数可以以符号或者拓扑地址的方式输入和显示
- 检查窗口的显示

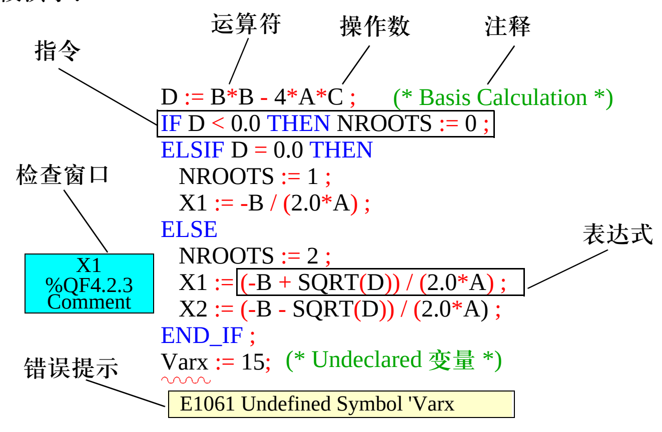
结构化文本 ST

介绍

ST 编辑器用于符合 IEC 61131-3 的结构化文本编程。

演示

ST 代码段演示：



对象

ST 编程语言通过 “表达式”进行操作。

表达式由运算符和操作数构成，它在执行的时候会返回一个数值。

运算符是在执行运算过程中所用到的符号。

运算符用于操作数。操作数包括变量，字面值，功能以及功能块输入 / 输出，等等。

指令用来构建和控制表达式。

输入辅助帮助

ST 编辑器提供了以下输入帮助：

- 在创建程序的时候，直接进行语法和语义检查
  - 用颜色显示出关键字和注释
  - 未知字 (比如未经声明的变量) 或者不匹配的数据类型会用红色的波浪线标出
  - 在快速信息 (错误提示) 中给出简短错误说明
- 表格显示功能和功能块
- 功能和功能块的输入辅助功能
- 操作数可以以符号或者拓扑地址的方式输入和显示
- 检查窗口显示

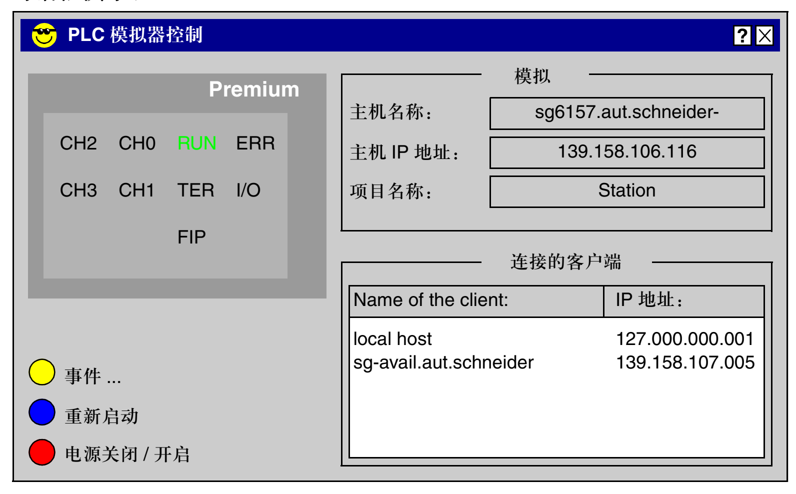
PLC 模拟器

介绍

通过 PLC 模拟器，用户可以不必连接到真实 PLC 的情况下进行程序调试。
在真实 PLC 上运行的所有项目任务 (Mast， Fast， AUX 和 Event)，都可以在模拟器上运行。该模拟器和真实 PLC 的区别在于它没有 I/O 模块和通信网络 (比如：ETHWAY， Fipio 和 Modbus Plus) 的非确定性实时行为。
PLC 模拟器也具有所有调试功能，动态仿真，断点，强制变量等内容。

演示

对话框演示：



模拟器的结构

模拟器控制器提供了以下可视内容：
- 模拟的 PLC 的类型
- 模拟的 PLC 的当前状态
- 加载项目的名称
- 用于模拟器的主机 IP 地址和 DNS 名称，以及所有建立连接的客户机
- 用来模拟 I/O 事件的对话框
- 用来重启模拟 PLC 的重新启动按钮 (模拟的冷启动)
- 用来模拟热重启的电源关闭 / 开启按钮
- 用来控制模拟器的快捷菜单 (鼠标右键)

导出 / 导入

介绍

导入和导出功能允许用户在新项目中使用已有的数据。借助 XML 导出 / 导入格式，用户可以向 / 从外部软件提供 / 接受数据。

导出

以下对象可以被导出：
- 包括配置在内的完整的项目
- 所有编程语言的代码段
- 所有编程语言的子程序代码段
- 导出功能块 (DFB)
- 导出数据类型 (DDT)
- 变量声明
- 操作屏
- 配置 (FNES 格式)

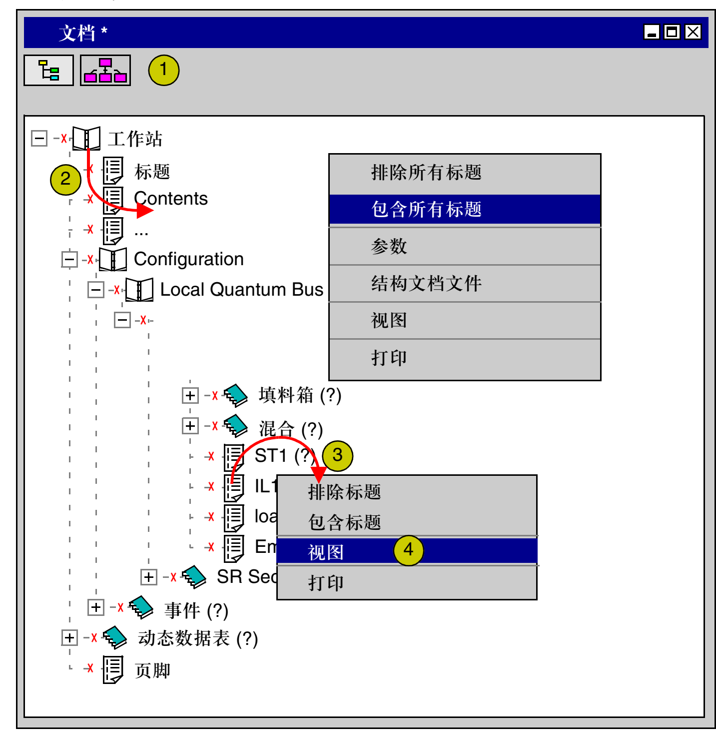
导入

所有能够导出的对象也可以被导入。
有两种导入方式：
- **直接导入**
  完全按照导出时的情况将对象导入。
- **通过辅助功能进行导入**
  辅助功能允许用户更改变量名称，代码段或者功能组件。用户也可以修改地址的映射。

用户文档

用户文档

用户文档范围：



以下是针对项目文档操作所提供的一些服务功能：

- 对整个项目 (2) 或者按代码段进行打印 (3)
- 在结构和功能视图之间进行选择 (1)
- 调整结果 (页脚，常规信息，等等。)
- 为编程语言编辑器，配置器等进行本地打印
- 突出显示 (粗体) 关键字
- 可以选择文件格式
- 打印预览 (4)
- 保存文档

调试服务

在用户应用程序中寻找错误

以下功能用来在项目中进行调试优化：

- 在编程语言编辑器中设置断点
- 逐步执行程序，包括进入，跳出和跨过每一步
- 调用内存，以便恢复整个程序路径
- 控制输入和输出

联机模式

联机模式指个人计算机和 PLC 建立了连接的情况。用户可以在 PLC 上使用联机模式来进行调试，动态操作以及更改程序。

在个人计算机和 PLC 建立了连接以后，系统会自动对个人计算机的项目和 PLC 的项目进行比较。

比较可能有以下几种结果：

- **个人计算机和 PLC 上的项目不同：**
  在这种情况下，联机模式会被限制。用户只能使用 PLC 控制命令 (比如开始，中止)，诊断服务和变量监视。用户不能更改 PLC 程序逻辑或者配置。不过，下载和上传功能是可用的，可以在不受限制的模式下运行 (个人计算机和 PLC 上的项目相同)。
- **个人计算机和 PLC 上的项目相同：**
  有两种可能性：
  - 联机相同，创建的
    上一次在个人计算机上所创建的项目被下载到 PLC 上，此后没有发生更改，也就是说，个人计算机上的项目和 PLC 上的是完全一致的。
    在这种情况下，所有动态功能都是可用的，不受限制的。
  - 联机相当，未创建的
    上一次在个人计算机上所创建的项目被下载到 PLC 上，不过后来发生了改变。
    在这种情况下，动态功能只能用于未改动过的项目组件中。

动态化

变量的动态化有各种可能性：

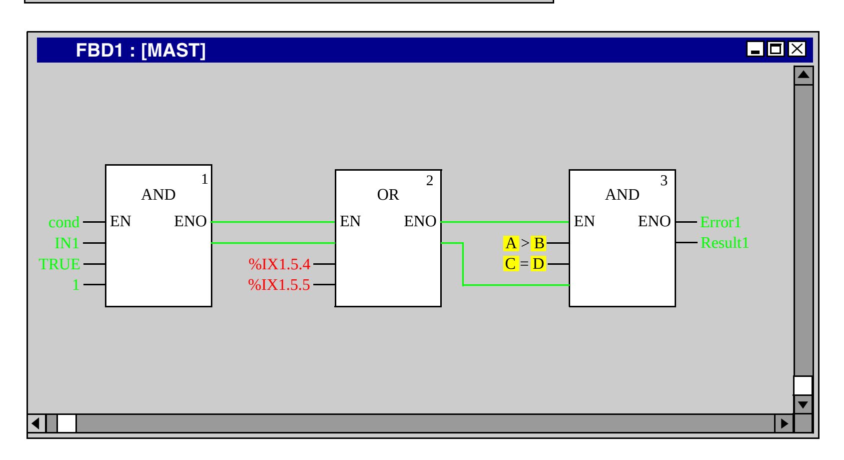
- **代码段动态化**

  所有编程语言 (FBD，LD，SFC，IL 和 ST) 都可以动态化。变量在连接时可以直接在代码段中动态化。

ST1 : [MAST]

```
   TIMER(IN := NOT pulse;
         PT := t#1s;  (* Blink timer *)
pulse := TIMER.Q;

(* Count every pulse *)
IF pulse = 1 THEN
   count := count + 1;
END_IF;
(* Animate lights according to counter *)
CASE count OF
   1: out1 := TRUE;
   2: out2 := TRUE;
ELSE (* All lights are on, switch then off again and start from
   out1 := FALSE;
   out2 := FALSE;
```



- **工具提示**
  带有一个变量值的工具提示会在鼠标掠过该变量区域的时候显示出来。

ST1 : [MAST]

```
   TIMER(IN := NOT pulse;
            PT := t#1s;  (* Blink timer *)
pulse := TIMER.Q;

(* Count every pulse *)
IF pulse = 1 THEN
   count := count + 1;
END_IF;        66
(* Animate lights according to counter *)
CASE count OF
   1: out1 := TRUE;
   2: out2 := TRUE;
ELSE (* All lights are on, switch then off again and start from
   out1 := FALSE;
   out2 := FALSE;
```

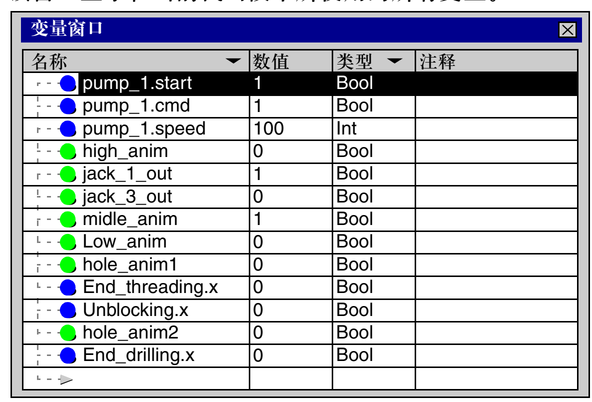
- **检查窗口**
  用户可以为任何变量创建一个检查窗口。该窗口显示变量值，地址和所有注释 (如果有的话)。在所有编程语言中，该功能都可用。

ST1 : [MAST]

```
   TIMER(IN := NOT pulse;
            PT := t#1s;  (* Blink timer *)
pulse := TIMER.Q;

(* Count every pulse *)
IF pulse = 1 THEN         count
   count := count + 1;      7
END_IF;
(* Animate lights according to counter *)
CASE count OF
   1: out1 := TRUE;     out1
   2: out2 := TRUE;     TRUE
ELSE (* All lights are on, switch then off again and start from
   out1 := FALSE;
   out2 := FALSE;
```

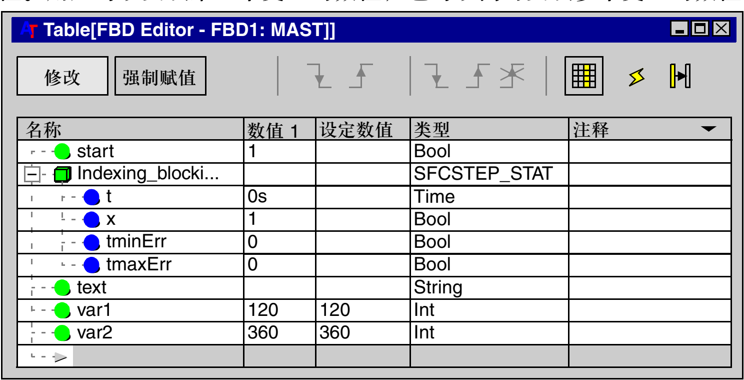
- **变量窗口**
  该窗口显示在当前代码段中所使用的所有变量。



变量窗口

名称	数值	类型	注释
pump_1.start	1	Bool	
pump_1.cmd	1	Bool	
pump_1.speed	100	Int	
high_anim	0	Bool	
jack_1_out	1	Bool	
jack_3_out	0	Bool	
midle_anim	1	Bool	
Low_anim	0	Bool	
hole_anim1	0	Bool	
End_threading.x	0	Bool	
Unblocking.x	0	Bool	
hole_anim2	0	Bool	
End_drilling.x	0	Bool	

- **动态数据表**
  用户可以在动态数据表中显示，更改任何变量的数值，或者对它们进行强制赋值。用户可以更改某一个变量的数值，也可以同时更改多个变量的数值。



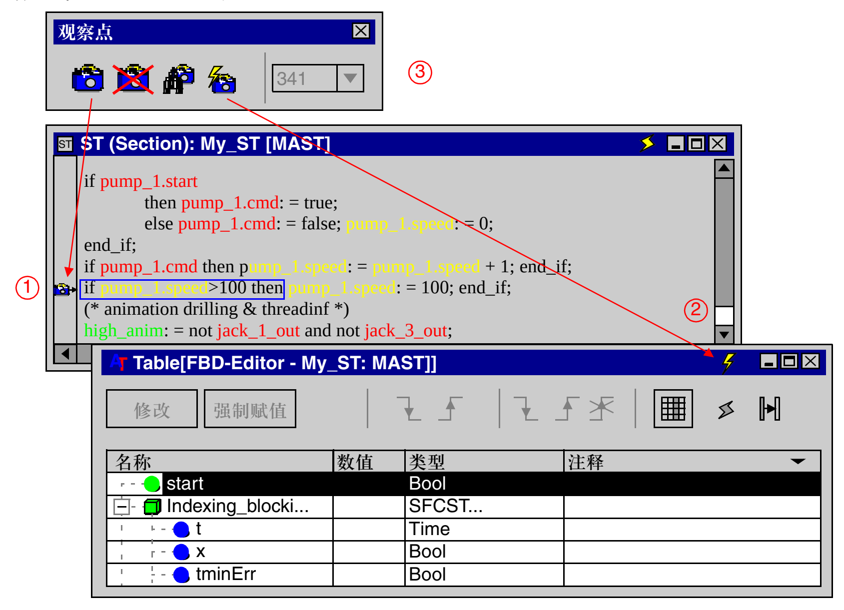
Table[FBD Editor - FBD1: MAST]]

修改 强制赋值

名称	数值 1	设定数值	类型	注释
start	1		Bool	
Indexing_blocki...			SFCSTEP_STAT	
t	0s		Time	
x	1		Bool	
tminErr	0		Bool	
tmaxErr	0		Bool	
text			String	
var1	120	120	Int	
var2	360	360	Int	

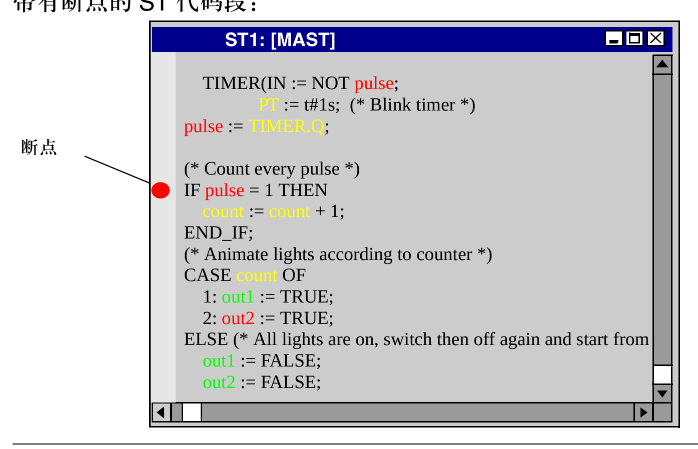
观察点

用户可以在创建观察点 (1) 的同时，借助这些观察点浏览 PLC 数据，而不必等到周期结束。
动态数据表可以和观察点 (2) 同步。
观察点更新的频率由一个计数器 (3) 来决定。
带有观察点的 ST 代码段：



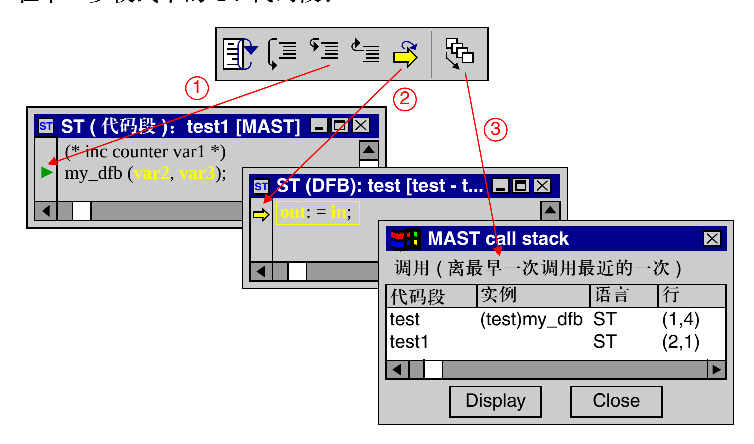
断点

借助断点，用户可以在项目的任意位置停止处理过程。
带有断点的 ST 代码段：



单一步模式

单一步模式允许用户逐步执行程序。如果项目被某个断点所中止，或者已经处于单一步模式，那么就会提供单一步功能。
在单一步模式下的 ST 代码段：



在单一步模式中提供了以下功能：
- 逐步执行程序
- 进入步 (1)
- 跳出步
- 跨过步
- 显示当前步 (2)
- 调用内存 (3)

  在 “步信息” 功能被执行若干次以后，调用内存功能就会从第一个断点处起显示整个路径。

书签

借助书签，用户可以对代码段进行选择，并且很容易地再次找到它们。

诊断浏览器

描述

Unity Pro 提供了系统和项目诊断功能。

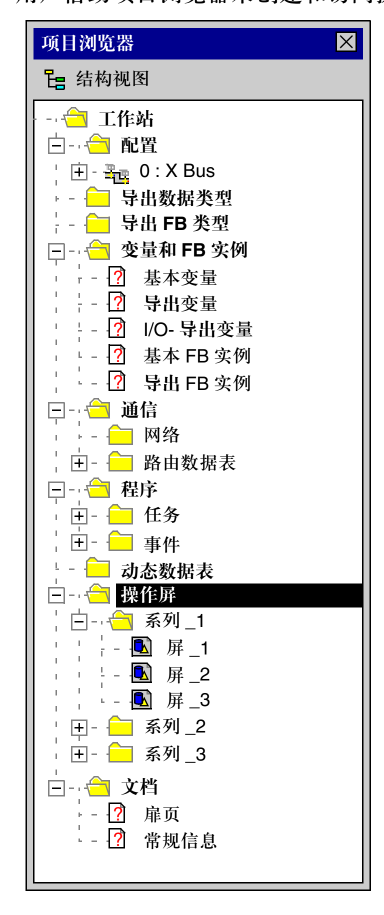
发生的错误会在一个诊断窗口内显示出来。用户可以直接从诊断窗口打开引起错误的代码段，以便于纠正错误。



操作屏

介绍

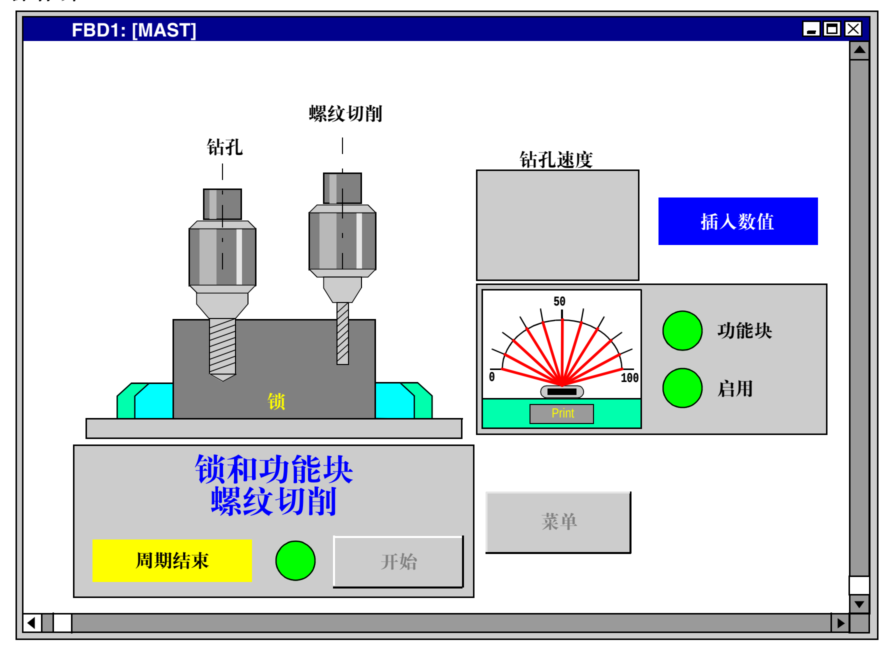
操作员窗口显示自动化进程。
借助操作屏编辑器，用户可以很方便地创建，修改和管理操作屏。
用户借助项目浏览器来创建和访问操作屏。



操作屏编辑器

操作员窗口中含有很多信息 (动态变量，系统概览，文本说明，等等)，使得用户能够方便地监视和更改自动化变量。

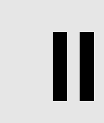
操作屏



操作屏编辑器提供了如下功能：

- 丰富的显示功能
  - 几何元素
    线，矩形，椭圆，曲线，多边形，位图，文本
  - 控制元素
    按钮，控制箱，开关，页面浏览，超链接，输入区，旋转区
  - 动态元素
    棒状图，趋势图，对话，日期，消失，闪烁色，变量动态化
- 为管理图形对象创建一个库
- 复制对象
- 创建一个在操作屏中使用的所有变量的列表
- 创建在操作屏中使用的消息
- 从操作屏直接访问动态数据表或者交叉引用数据表的一个或多个变量
- 工具提示给出关于变量的附加信息
- 按照系列来管理操作屏
- 导入 / 导出个体操作屏或整个系列

应用程序结构



内容预览

本部分内容

本部分描述了与每种 PLC 相关的应用程序和内存结构。

本部分内容

本部分包含以下各章：

每种 PLC 可用功能的描述

2

各种 PLC 的可用功能

编程语言

可用语言

平台	Premium: TSX			Atrium: TSX	Quantum: 140 CPU	
处理器	P57 1••	P57 2•• P57 3•• P57 4••	P57 5••	PCI 57 204/ 454	31•••• 43•••• 53••••	651•• 671••
LD 语言	X	X	X	X	X	X
FBD 语言	-	X	X	X	X	X
ST 语言	X	X	X	X	X	X
IL 语言	X	X	X	X	X	X
SFC 语言	X	X	X	X	X	X
图例说明						
X	可用语言。					
-	不可用语言。					

任务和处理

可用的任务和处理

平台	**Premium: TSX**			**Atrium: TSX**	**Quantum: 140 CPU**	
处理器	P57 1••	P57 2•• P57 3•• P57 4••	P57 5••	PCI 57 204/ 454	31•••• 43•••• 53••••	651•• 671••
主任务 循环式或周期式	X	X	X	X	X	X
快速任务 周期式	X	X	X	X	X	X
辅助任务 周期式	-	-	4	-	-	4
I/O 类型事件处理	32	64	128	64	64	128
计时器类型事件处理	-	-	32	-	16	32
I/O 类型和计时器类型事件处理总计	32	64	128	64	64	128
图例说明						
X 或数值	可用的任务或进程。该数值给出了任务或处理的最大数量。					
-	不可用的任务或处理。					

应用程序结构

3

内容预览

本章主题

本章描述了通过 Unity Pro 软件创建的程序的结构和执行情况。

本章内容

本章包含以下各节：

3.1 任务和进程描述

内容预览

本节主题

本节描述了包含应用程序的任务和进程。

本节内容

本节包含以下主题：

主任务介绍

常规信息

主任务是应用程序的主要任务。它是必需的，以缺省的方式创建。

结构

主任务 (MAST) 由代码段和子程序组成。

主任务的每一个代码段都用以下语言进行编写：LD， FBD， IL， ST 或者 SFC。

子程序用 LD， FBD， IL 或者 ST 进行编写，在任务代码段中进行调用。

注意：SFC 只能用在主任务代码段。用 SFC 编写的代码段的数量不受限制。

执行

用户可以选择执行主任务的类型：
- 循环式 (缺省的选择)
- 或者周期式 (1 到 255 毫秒)

注意：如果主任务是循环式的，那么辅助任务是被禁止的。

控制

主任务可以用程序通过位和系统字来控制。

系统对象	描述
%SW0	任务周期
%S30	主任务激活
%S11	监视时钟错误
%S19	周期溢出
%SW30	上个周期的执行时间 (毫秒)
%SW31	最长周期的执行时间 (毫秒)
%SW32	最短周期的执行时间 (毫秒)

快速任务介绍

常规信息

快速任务用于持续时间较短的、周期性的处理任务。

结构

快速任务 (FAST) 由代码段和子程序组成。

快速任务的每一个代码段都是用以下语言中的一种编写的：LD， FBD， IL 或者 ST。

在快速任务的代码段中不能使用 SFC 语言。

子程序是用 LD， FBD， IL 或者 ST 语言编写的，并在任务代码段中调用。

执行

快速任务的执行是周期式的。
它的优先级比主任务高。

快速任务 (FAST) 的周期由配置来确定，其数值从 1 到 255 ms。

不过，被执行的程序的时间必须尽量短，以避免低优先级任务的溢出。

控制

快速任务可以用程序通过字或系统位来控制。

系统对象	描述
%SW1	任务周期
%S31	快速任务激活
%S11	监视时钟错误
%S19	周期溢出
%SW33	上个周期的执行时间 (ms)
%SW34	最长周期的执行时间 (ms)
%SW35	最短周期的执行时间 (ms)

辅助任务介绍

常规信息

辅助任务用于较慢的处理任务。它们是优先级最低的任务。

在 Premium TSX P57 5•• 和 Quantum 140 CPU 6•••• PLC 上最多可以编写 4 个辅助任务 (AUX0， AUX1， AUX2 或 AUX3)

结构

辅助任务 (AUX) 由代码段和子程序组成。

每一个辅助任务的代码段都是用以下几种语言之一编写的：LD， FBD， IL 或者 ST。

在辅助任务的代码段中不能使用 SFC 语言。

用 LD， FBD， IL 或者 ST 语言最多可以编写 64 个子程序。这些子程序由任务代码段进行调用。

执行

辅助任务的执行是周期式的。

它们的优先级是最低的。

辅助任务的周期可以固定在 10 ms 到 2.55 s 之间。

控制

辅助任务可以用程序通过位和系统字来控制。

系统对象	描述
%SW2	辅助任务 0 的周期
%SW3	辅助任务 1 的周期
%SW4	辅助任务 2 的周期
%SW5	辅助任务 3 的周期
%S32	激活辅助任务 0
%S33	激活辅助任务 1
%S34	激活辅助任务 2
%S35	激活辅助任务 3
%S11	监视时钟错误
%S19	周期溢出
%SW36	辅助任务 0 上个周期的执行时间 (ms)
%SW39	辅助任务 1 上个周期的执行时间 (ms)
%SW42	辅助任务 2 上个周期的执行时间 (ms)
%SW45	辅助任务 3 上个周期的执行时间 (ms)
%SW37	辅助任务 0 最长周期的执行时间 (ms)
%SW40	辅助任务 1 最长周期的执行时间 (ms)
%SW43	辅助任务 2 最长周期的执行时间 (ms)
%SW46	辅助任务 3 最长周期的执行时间 (ms)
%SW38	辅助任务 0 最短周期的执行时间 (ms)
%SW41	辅助任务 1 最短周期的执行时间 (ms)
%SW44	辅助任务 2 最短周期的执行时间 (ms)
%SW47	辅助任务 3 最短周期的执行时间 (ms)

事件处理概述

常规信息

事件处理用来缩短下列事件应用程序的响应时间：
- 来自输入 / 输出模块，
- 来自事件计时器。

这些处理任务的优先级高于任何其他任务。所以，它们适用于需要很短的事件响应时间的处理任务。

能够编写的事件处理任务的数量 (参见*任务和处理，64 页*) 取决于处理器的类型。

结构

事件处理任务是单代码段的，它只含有一个 (无条件的) 代码段。

它是用 LD， FBD， IL 或 ST 语言中的一种来编写的。

有两种事件：
- EVTi：用于来自输入 / 输出模块的事件
- TIMERi：用于来自事件计时器的事件

执行

事件处理任务的执行是异步的。

在出现了某个事件以后，系统会重新安排应用程序到输入 / 输出通道的路径。

控制

在执行程序的过程中，用户可以通过如下系统位和字来控制事件处理任务。

系统对象	描述
%S38	事件处理激活
%S39	事件调用管理堆栈饱和
%SW48	执行事件处理任务的数量

3.2 代码段和子程序描述

内容预览

本节主旨 本节描述了构成任务的代码段和子程序。

本节内容 本节包含以下主题：

代码段描述

代码段概述

代码段是自治的编程实体。

指令行的识别标号和接点网络等是每个代码段所专有的内容 (不能和别的代码段进行程序跳转)。

它们用下列语言中的一种来编写：
- 梯形图语言 (LD)
- 功能块语言 (FBD)
- 指令表 (IL)
- 结构化文本 (ST)
- 或者顺序功能图 (SFC)

编写的前提是相关的任务能够接受该语言。

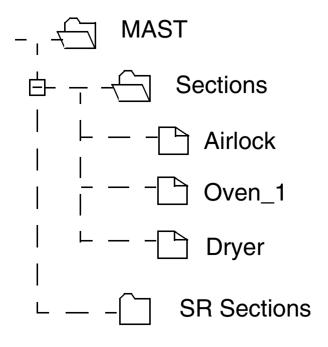
代码段按照它们在浏览器窗口内的编写顺序来执行 (结构视图)。

在主任务，快速任务和辅助任务中，可以把一个执行条件和一个或多个代码段相关联，但不能用在事件处理任务中。

代码段与一个任务相链接。同一个代码段不能同时和多个任务相链接。

例子

下面的图表显示了一个组织成代码段的任务。



代码段的特性

下面的表格描述了代码段的特性。

特性	描述
名称	最多为 32 个字符 (可以使用撇号，但是不能使用空格)。
语言	LD， FBD， IL， ST 或者 SFC
任务或处理	主任务，快速任务，辅助任务，事件
条件 (可选)	可以用一个 BOOL 或者 EBOOL 类的位变量来为代码段的执行设置条件。
注释	最多为 256 个字符。
保护	写保护，读 / 写保护。

SFC 代码段描述

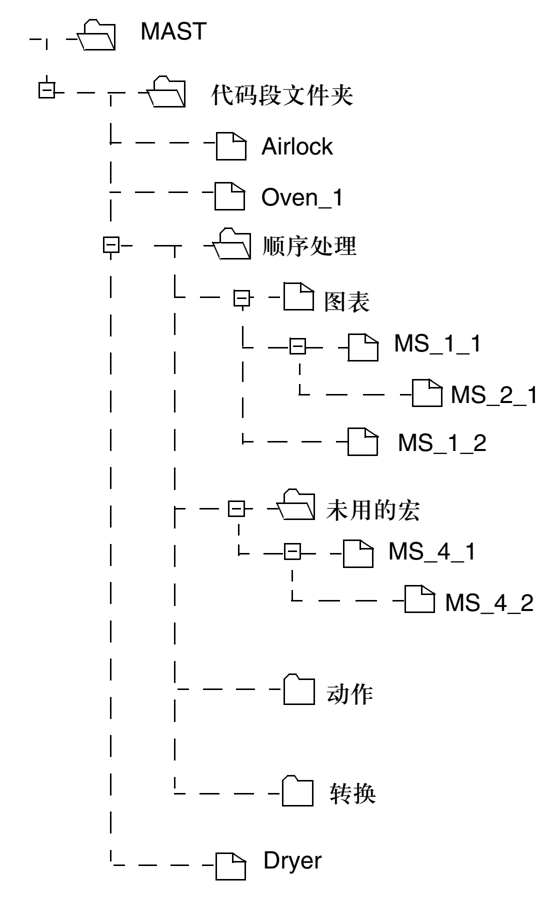
常规信息

在顺序功能图语言中的代码段由以下内容组成：
- 一个用 SFC 编写的主图表
- 用 SFC 编写的宏步 (MS)
- 用 LD， FBD， ST，或者 IL 编写的动作或转换。

SFC 代码段只能在主任务中编写 (请参见关于 SFC 代码段的详细描述)

例子

下面的图表给出了一个 SFC 代码段的结构实例，并借助相关的图表显示了它所使用的宏步调用。



子程序描述

子程序概述

子程序作为单独的实体，使用以下几种语言之一进行编写：

- 梯形图语言 (LD)，
- 功能块语言 (FBD)，
- 指令表 (IL)，
- 结构化文本 (ST)。

子程序可以在代码段或者其他子程序中来调用。嵌套的数量最多为 8 个。
子程序不能调用自身 (非递归)。

子程序也与一个任务相链接。同一个子程序不能被多个不同的任务所调用。

例子

下面的图表显示了组织成代码段和子程序的任务。

MAST
代码段
SR 代码段
Control_1
Detection

子程序的特性

下面的表格描述了子程序的特性。

特性	描述
名称	最多为 32 个字符 (可以使用下横杠，但是不能使用空格)
语言	LD， FBD， IL 或者 ST.
任务	主任务，快速任务或者辅助任务
注释	最多为 512 个字符

3.3 单任务执行

内容预览

本节主题

本节描述了单任务应用程序的操作。

本节内容

本节包含以下主题：

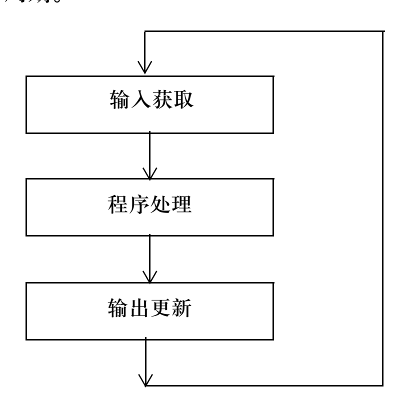
主任务周期描述

常规信息

单任务应用程序是指只有一个用户任务，也就是主任务 (参见*主任务介绍，67 页*)。
您可以选择执行主任务的类型：
- 循环式
- 或者周期式

图例

下面的图例显示了操作周期。



不同阶段的描述

下面的表格描述了各个操作阶段。

阶段	描述
输入获取	把与任务相关联的离散量模块和专用模块的输入数据状态写到内存中去，用户可以通过强制赋值操作修改它们的值。
程序处理	执行由用户编写的应用程序
输出更新	根据应用程序所执行的结果，把输出位或字写到与任务相关联的离散量模块和专用的模块中去。 用户可以通过强制赋值的操作对输出的数值进行修改。

注意：在输入获取和输出更新阶段，系统还会隐式地监视 PLC (系统位和字的管理，实时时钟当前值的更新，状态发光二极管和液晶显示屏的更新，运行 / 停止模式转换检测，等等) 和处理来自终端的请求 (修改和动态化)。

操作模式

运行的 PLC，处理器按照顺序进行内部处理，输入获取，应用程序处理以及输出更新。

停止的 PLC，处理器进行以下操作：

- 内部处理，
- 输入获取 (1)，
- 根据所选择的配置：
  - 返回模式：输出设置到返回值。
  - 保持模式：保留上一个输出值。

(1) 适用于 Premium 和 Atrium PLCs ；对于 Quantum PLC 来说，当 PLC 处于停止状态时，禁止进行输入获取。

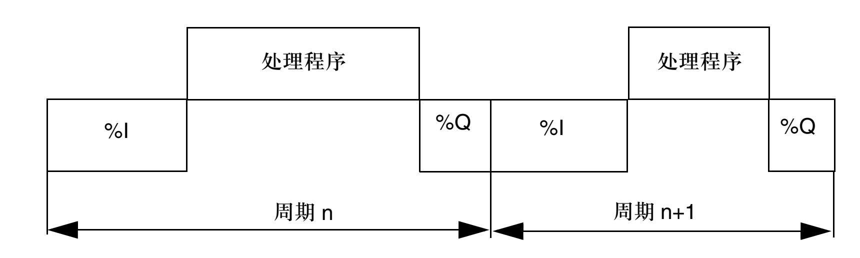
单任务：循环式执行

常规信息

主任务按照下面所述的方式进行操作。此处描述了在单任务操作中循环执行主任务的过程。

操作

下图显示了 PLC 周期的各个执行阶段。



%I 读输入

%Q 写输出

描述

这种操作由连续的周期所构成。

在完成了输出更新以后，系统会执行自己的特殊处理操作，然后不间断地开始一个新的周期。

周期检查

监视时钟会对周期进行检查 (参见*扫描时间控制，82 页*)。

周期式执行

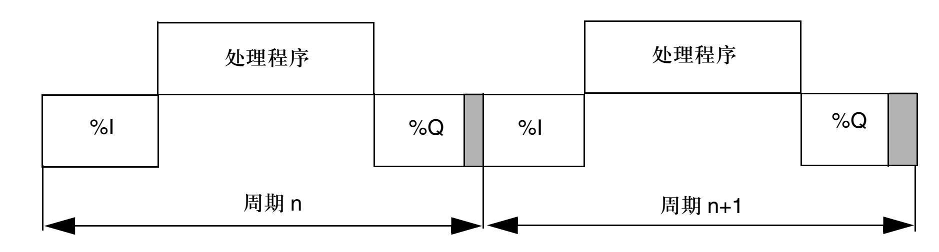
描述

在这种操作模式下，输入获取，应用程序处理和输出更新会根据一个定义的扫描时间周期执行，这个定义的扫描时间为 1 到 255 ms。

在每个 PLC 周期的开始阶段，一个根据定义的扫描时间对当前定时器进行初始化，并启动倒计时。
在定时器时间用完以前， PLC 周期必须结束，以启动一个新的周期。

操作

下面的图表显示了 PLC 周期的各个执行阶段。



%I 读输入

%Q 写输出

操作模式

处理器按照顺序进行内部处理，输入获取，应用程序处理以及输出更新。
- 如果扫描时间尚未结束，处理器会通过进行内部处理操作来完成操作周期，直到扫描时间耗尽。
- 如果操作时间大于分配的扫描时间，PLC会通过将任务的系统位%S19设置为1而发出溢出信号。接下来系统就会继续处理过程，并完成操作 (不过，它必须不能超过监视时钟的时间限定)。在输出被当前周期明确写入以后，下一个周期就会开始。

周期检查

有两种检查：
- 周期溢出 (参见*扫描时间控制，82 页*)，
- 通过监视时钟 (参见*扫描时间控制，82 页*)，

扫描时间控制

常规信息

在循环式或者周期式操作中，主任务的执行周期被 PLC (监视时钟) 所控制，不能超过在 Tmax 配置中定义的数值 (缺省值为 250 ms，最大值为 2.55 s)。

监视时钟 (周期式或者循环式操作)

如果监视时钟发生了溢出，系统就会报错，从而使 PLC 暂时停止 (暂停)。

%S11 位表示监视时钟的溢出。当扫描时间大于监视时钟时，系统就会将其设定为 1。

字 %SW11 含有以毫秒为单位的监视时钟值。用户不能通过程序更改这个数值。

注意：
- 如果要再次激活任务，需要把终端连上，以便于分析错误原因，纠正错误，对 PLC 重新进行初始化，并将其切换到运行模式。
- 如果要退出暂停模式，仅仅通过将其切换到停止模式是不行的，用户必须对应用程序重新进行初始化，以确保数据的一致性。

周期性操作中的控制

在周期性操作中，用户可以通过一个附加的控制功能来检测周期溢出。如果周期溢出的时间小于监视软件的值，那么周期溢出就不会使 PLC 停止。

%S19 位表示周期溢出。当扫描时间大于任务周期的时候，系统会把它的值设定为 1。

%SW0 字含有周期值 (以毫秒为单位)，这个值在冷重启的时候由定义值进行初始化。用户可以更改这个数值。

主任务执行时间的利用

以下系统字可以用来获取与扫描时间相关的信息：
- %SW30 包含上一个周期的执行时间，
- %SW31 包含最长周期的执行时间，
- %SW32 包含最短周期的执行时间。

注意：用户也可以通过配置编辑器来访问这些信息项。

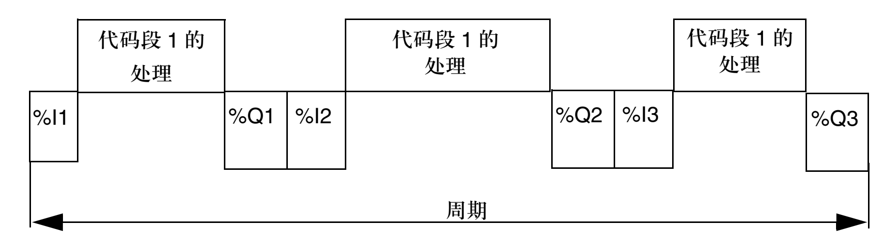
带有分布式输入 / 输出的 Quantum 代码段的执行

常规信息

Quantum PLC 有一个专用的代码段管理系统。它适用于带有远程输入 / 输出的站。系统允许代码段以优化的响应时间，对输入 / 输出进行更新 (在更新输入 / 输出时，不需要等待整个任务周期)。

操作

下面的图表显示了一个带有 3 个代码段的任务的各个执行阶段，这 3 个代码段与 3 个远程输入 / 输出站相关联。



%Ii 读 i 号站的输入

%Qi 写 i 号站的输出

描述

阶段	描述
1	读入与 1 号代码段相关联的远程输入 / 输出站的输入。
2	处理代码段 1 的程序。
3	读入与 1 号代码段相关联的远程输入 / 输出站的输出。
4	周期将以同样的方式继续执行后面的代码段和远程站。

3.4 多任务执行

内容预览

本节主题

本节针对多任务应用程序的操作进行了描述。

本节内容

本节包含以下主题：

多任务软件结构

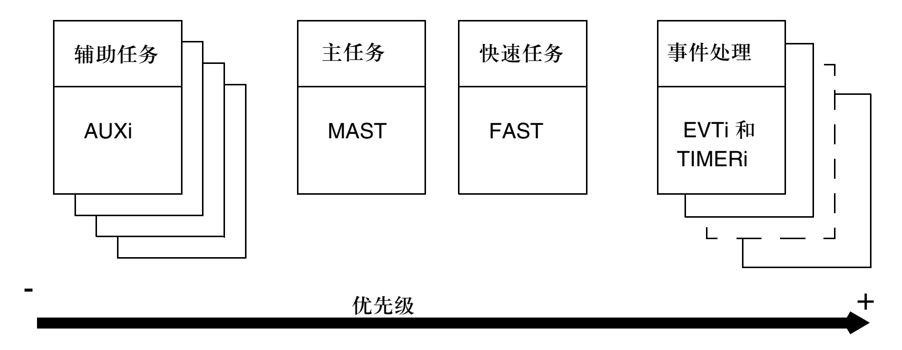
任务和处理

这种应用程序的任务结构如下所示：

任务 / 处理	名称	描述
主任务	MAST	在任何情况下都存在，可以是循环式的或者周期式的。
快速任务	FAST	可选，总是周期式的。
辅助任务	AUX 0 到 3	可选，总是周期式的。
事件	EVTi 和 TIMERi (参见 *常规信息， 92 页*)	当某个事件在一个输入 / 输出模块上发生，或者被事件计时器所触发，系统就会调用它。 这些处理类型是可选的，可以被那些要在较短响应时间内作用于输入 / 输出的应用程序所使用。

图例

以下图表显示了多任务结构中的任务，以及它们的优先级。



描述

主 (MAST) 任务仍然是应用程序的基础。其他任务则根据 PLC 的类型而有所不同 (参见 *任务和处理， 64 页*)。
每一个任务的优先级都是固定的，以便于用户能够在进行特定处理的过程中区分任务的优先次序。

通过外部事件生成的命令可以通过与周期式任务不同步的方式激活事件处理。系统会优先处理该命令，并停止所有正在处理的任务。

在多任务结构中的任务顺序

常规信息

在缺省的情况下，主任务处于激活状态。
Fast 和辅助任务如果事先被编好，那么在缺省的情况下，它们也处于激活状态。事件处理则在发生了相关事件的情况下被激活。

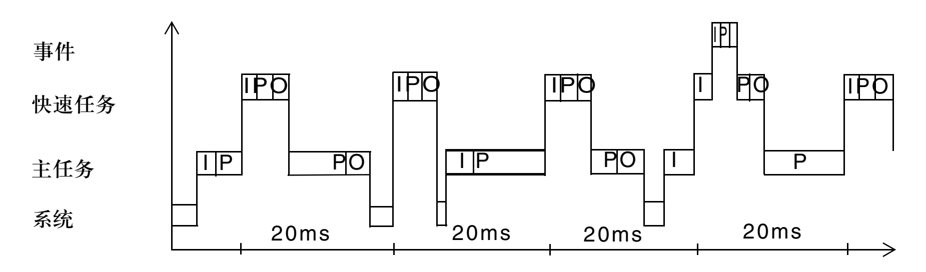
操作

下面的表格给出了具有高优先级的任务的执行情况 (下面的图表也演示了这一操作)。

阶段	描述
1	某一个事件发生，或者快速任务周期开始。
2	停止执行低优先级任务。
3	执行高优先级的任务。
4	在高优先级任务执行完毕以后，重新恢复中断的任务。

任务顺序描述

下面的图表演示了多任务处理的顺序，其中包括一个循环式主任务，一个具有 20 毫秒周期的快速任务，以及事件处理。



图例说明：
I：输入获取
P：程序处理
O：输出更新

任务控制

用户可以通过程序，借助系统位控制快速任务和事件处理任务的执行：
- %S30 用来控制 MAST 主任务是否被激活。
- %S31 用来控制 FAST 任务是否被激活。
- %S32 到 %S35 用来控制辅助任务 AUX0 到 AUX3 是否被激活。
- %S38 用来控制 EVTi 事件处理是否被激活。

注意：基本功能 MASKEVT 和 UNMASKEVT 也允许程序对事件进行全局掩码和消除掩码操作。

任务控制

循环式和周期式操作

在多任务处理中，具有最高优先级的任务应该使用周期式模式，以便具有较低优先级的任务能够拥有足够的执行时间。

出于这个原因，只有具有最低优先级的任务才应该使用循环模式。因而，如果为主任务选择了循环操作模式，那么就不能使用辅助任务。

任务持续时间的测量

系统会不断测量任务的持续时间。这种测量会给出任务执行开始和结束之间的持续时间。

这种测量包含由较高优先级的任务所占有的时间，这些任务可能会中断正在测量的任务的执行过程。

以下系统字给出了每一个任务的当前，最大和最小扫描时间 (以 ms 为单位)

时间测量	MAST	FAST	AUX0	AUX1	AUX2	AUX3
当前	%SW30	%SW33	%SW36	%SW39	%SW42	%SW45
最大	%SW31	%SW34	%SW37	%SW40	%SW43	%SW46
最小	%SW32	%SW35	%SW38	%SW41	%SW44	%SW47

注意： 最大和最小时间来自上一次冷重启以后测得的时间。

任务周期

任务周期在任务属性中进行定义。它们可以被以下系统字所修改。

系统字	任务	数值	缺省值	注释
%SW0	MAST	0..255ms	循环式	0 = 循环式操作
%SW1	FAST	1..255ms	5ms	-
%SW2	AUX0	10ms..2.55s	100ms	扫描时间以 10 ms 来表示。
%SW3	AUX1	10ms..2.55s	200ms	
%SW4	AUX2	10ms..2.55s	300ms	
%SW5	AUX3	10ms..2.55s	400ms	

当任务的扫描时间超过了这个周期以后，系统会将任务的系统位 %S19 设为 1，并继续进行下一个扫描周期的操作。

注意： 周期的数值与任务的优先级没有关系。用户可以定义比主任务更长的快速任务周期。

监视时钟

通过任务属性，用户可以使用监视时钟来控制每一个任务的执行。
下面的表格给出了监视时钟针对每一种任务的取值范围：

任务	监视时钟的数值 (最小 ... 最大) (ms)	缺省的监视时钟的数值 (ms)	相关的系统字
MAST	10..1500	250	%SW11
FAST	10..500	100	-
AUX0	100..5000	2000	-
AUX1	100..5000	2000	-
AUX2	100..5000	2000	-
AUX3	100..5000	2000	-

如果监视时钟发生了溢出，系统就会报错，从而使 PLC 暂时停止 (暂停)。

%SW11 字包含主任务的监视时钟数值，该数值以毫秒为单位，它的数值不能被程序所修改。

%S11 位表示监视时钟的溢出。如果扫描时间比监视时钟更大，那么系统就会把它设定为 1。

注意：
- 如果要再次激活任务，需要把终端连上，以便于分析错误原因，纠正错误，对 PLC 重新进行初始化，并将其切换到运行模式。
- 如果要退出暂停模式，仅仅通过将其切换到停止模式是不行的，用户必须对应用程序重新进行初始化，以确保数据的一致性。

任务控制

在执行应用程序的过程中，用户可以通过以下系统位激活或者禁止某个任务：

系统位	任务
%S30	MAST
%S31	FAST
%S32	AUX0
%S33	AUX1
%S34	AUX2
%S35	AUX3

当相关的系统位被设定为 1 时，任务就会被激活。在主任务结束的时候，系统会对这些位进行检测。

如果某个任务被禁止，系统会连续读输入，写输出。

在启动应用程序的时候，在第一个执行周期，只有主任务会被激活。在第一个周期结束的时候，其他未被程序禁止 (相关的系统位被设为 0) 的任务会自动激活。

在读输入和写输出阶段的控制

以下系统字的位可以用来禁止读输入和写输出阶段。

禁止的阶段 ...	MAST	FAST	AUX0	AUX1	AUX2	AUX3
读输入	%SW8.0	%SW8.1	%SW8.2	%SW8.3	%SW8.4	%SW8.5
写输出	%SW9.0	%SW9.1	%SW9.2	%SW9.3	%SW9.4	%SW9.5

注意：
在缺省的情况下，读输入和写输出阶段是被激活的 (系统字的 %SW8 和 %SW9 位被设为 0)。
在 Quantum 上，通过 DIO 总线进行分配的输入 / 输出没有被 %SW8 和 %SW9 字所分配。

将输入 / 输出通道分配给主任务，快速任务和辅助任务

常规信息

每一个任务都会对分配给它的输入 / 输出进行写操作和读操作。

在通道，通道组，或者输入 / 输出模块与任务之间建立关联的操作，是在相应模块的配置页面进行定义的。
在缺省情况下的关联任务是 MAST 任务。

在 Premium 上的读输入和写输出

所有内置模块的输入 / 输出通道，都可以与一个任务 (MAST，FAST 或者 AUX 0..3) 建立关联。

本地和远程输入 / 输出 (X 总线)：
对于每一个任务周期来说，输入都在任务开始的时候被读入，输出都在任务结束的时候被写出。

在 Fipio 总线上的远程输入 / 输出：
在受控模式下，输入 / 输出的更新与任务周期相关。系统能够保证输入 / 输出在一个周期内被更新。只有与该任务相关的输入 / 输出会被更新。
在这种模式下， PLC 任务 (MAST，FAST or AUX) 的周期必须大于或者等于网络扫描时间。

在自由模式下，对任务周期没有限制。 PLC 任务周期 (MAST，FAST 或者 AUX) 可以小于网络周期。在这种情况下，在执行任务的时候不必更新输入 / 输出。如果选择了这种模式，您就可以为那些速度至为关键的任务获取最短的任务时间。

在 Quantum 上的读输入和写输出

本地输入 / 输出：
每一个输入 / 输出模块或者模块组都可以与一个任务 (MAST，FAST 或者 AUX 0..3) 进行关联。

远程输入 / 输出：
远程输入 / 输出工作站只能与主 (MAST) 任务相关联。代码段可进行相关的分配 (参见*带有分布式输入 / 输出的 Quantum 代码段的执行，83 页*)，每一个代码段可以与 31 个远程站中的一个站输入和 31 个远程输出站中的一个站输出相关联。

分布式输入 / 输出：
分布式输入 / 输出工作站只能与主 (MAST) 任务相关联。输入会在主任务开始的时候读入，输出会在主任务结束的时候写出。

Premium 上的例子

Premium 离散量模块的输入 / 输出可以按 8 个连续的通道为一组 (通道 0 到 7，通道 8 到 15，等等)，分配给 MAST， AUXi 或者 FAST 任务。

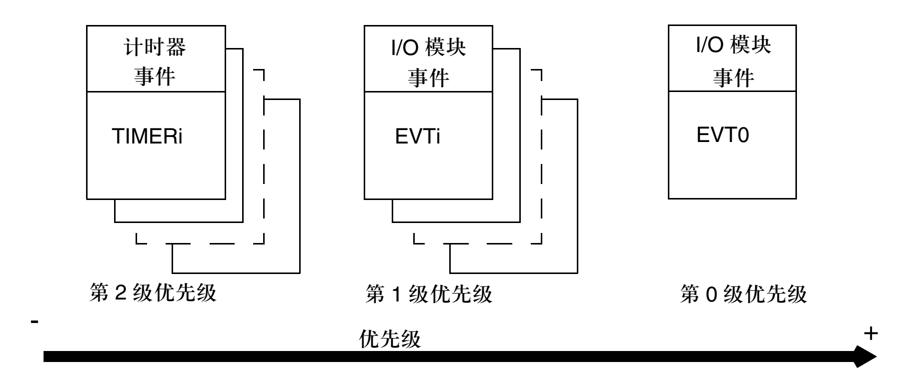
例子：可以将一个 28 输入 / 输出模块的通道进行如下分配：
- 把输入 0 到 7 分配给 MAST 任务，
- 把输入 8 到 15 分配给 FAST 任务，
- 把输出 0 到 7 分配给 MAST 任务，
- 把输出 8 到 15 分配给 AUX0 任务。

事件处理管理

常规信息

事件处理的优先级比其他任务高。

下面的图例说明了 3 种定义的优先级级别：



优先级管理

- EVT0 事件处理是最高的优先级处理。它可以中断其他类型的事件处理。
- 由输入/输出模块触发的EVTi事件处理的优先级(第1级优先级)比由计时器所触发的 TIMERi 事件处理 (第 2 级优先级) 高。
- 在 **Premium** 和 **Atrium PLC** 上：具有第 1 级优先级的事件处理会按照顺序进行存储和处理。
- 在 **Quantum PLC** 上：具有第 1 级优先级的处理类型，其优先级由以下因素决定:
  - 由机架上输入 / 输出模块的位置决定，
  - 由模块上通道的位置决定。

  位置号最小的模块具有最高的优先级。
- 由计时器所触发的事件处理具有第 2 级优先级。处理优先级是由最小的计时器号码决定的。

控制

通过系统位 %S38，应用程序可以激活或者禁止各种类型的事件处理。如果在禁止的过程中发生了一个或者多个事件，相关的处理就会丢失。

在应用程序中所使用的两种基本功能块 `MASKEVT()` 和 `UNMASKEVT()`，也可以用来对事件处理进行掩码或者消除掩码操作。

如果在对事件处理进行掩码操作的过程中发生了一个或多个事件，系统会把它们存储起来，在进行消除掩码操作以后会进行相关的处理。

计时器类型事件处理的执行

描述

在所有进程中，计时器类型事件处理都是由 ITCNTRL 功能触发的。这个计时器功能会在每次达到了预定值以后，周期性地激活事件处理。

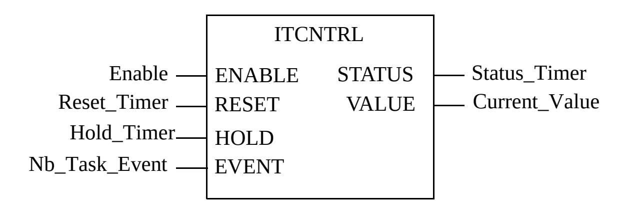
参考

在事件处理属性中可以选择如下参数。

参数	数值	缺省值	作用
时基	1 ms, 10ms, 100ms, 1 sec	10ms	计时器的时间基准。注意：应该谨慎地使用 1 毫秒的时基，因为如果处理触发频率太高，系统可能会发生溢出。
预设值	1..1023	10	计时器预设定数值。得到的时间周期等于：预设值 x 时基。
相位	0..1023	0	PLC 停止 / 运行切换时，第一次重新启动后，当前值是与计时器从 0 开始之间的偏移量数值。当前值等于： 相位 x 时基。

ITCNTRL 功能

FBD 中的演示：



下面的表格描述了输入参数：

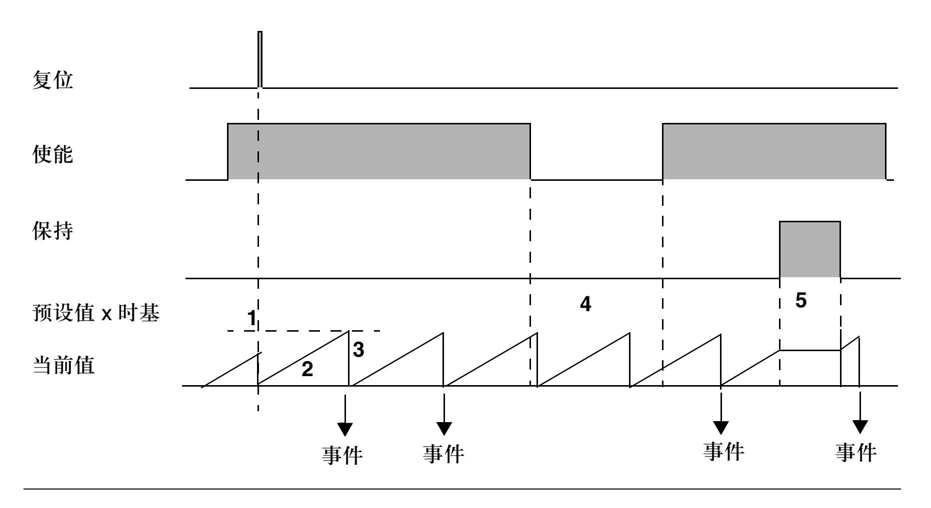
参数	类型	注释
Enable	BOOL	激活输入
Reset_Timer	BOOL	值为 1 时复位计时器
Hold_Timer	BOOL	值为 1 时保持计时器
Nb_Task_Event	BYTE	输入字节，确定要触发的事件处理号

下面的表格描述了输出参数：

参数	类型	注释
Status_Timer	WORD	状态字
Current_Value	TIME	计时器的当前值

正常操作的计时器时序图

计时器时序图



正常操作

下面的表格描述了计时器类型事件处理操作的触发 (参见上文的计时器图表)。

阶段	描述
1	当 RESET 输入接收到一个上升沿以后，计时器会复位为 0。
2	计时器的当前值 VALUE 从 0 向预设值增加，每个时基脉冲会使其值加一。
3	在当前值达到了预定值以后，就会产生一个事件，计时器会返回为 0，然后重新计时。如果事件没有被掩码，相关的事件处理也会触发。如果一个具有更高的或同等优先级别的事件处理任务正在执行，那么它就会延迟处理。
4	当 ENABLE 输入处于 0 的时候，计时器不会再计时，事件也就不会触发。
5	当 HOLD 输入处于 1 的时候，计时器被冻结，当前值停止增加，直到输入返回为 0。

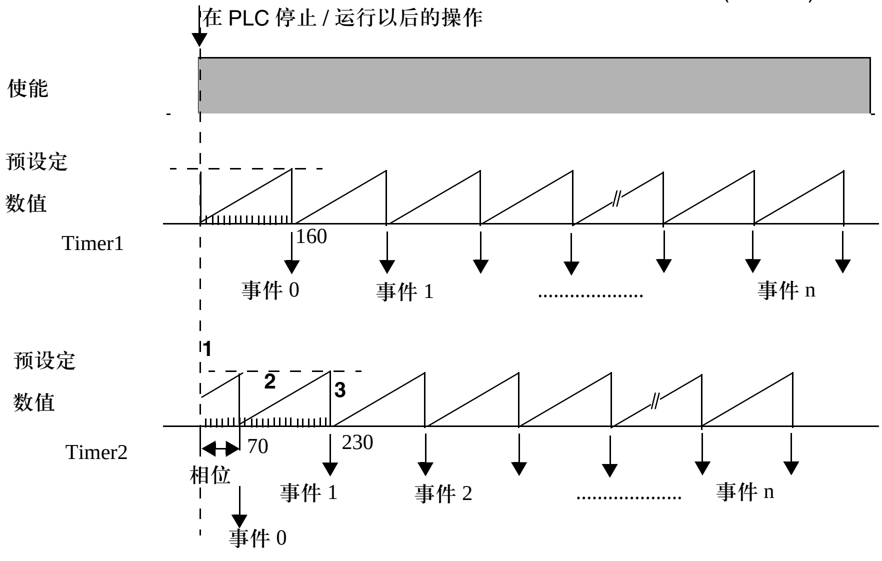
事件处理同步

相位参数用来定位触发不同计时器类型的事件处理任务。
这个参数设定为一个带有绝对时间原点的当前偏移值，绝对时间原点是 PLC 上一次从停止模式转换到运行模式而获得的。
操作条件：
- 事件处理任务必须具有同样的时基和预设值
- RESET 和 HOLD 输入不能被设定为 1。

例子：2 个事件处理任务 Timer1 和 Timer2 要以 70 毫秒的时间间隔执行。
第一个任务 Timer1 用相位 0 来进行定义，第二个任务 Timer2 用一个 70 ms 的相位进行定义 (相位 7，时间基准为 10 ms)。
任何触发 Timer1 相关任务的事件，都会在 70 ms 的时间间隔以后触发 Timer2 的相关任务。

时序图：停止 / 运行模式转换

上面例子的时序图，对 Timer1 和 Timer2 具有同一个预设定数值 16 (160 ms)。



在 PLC 停止 / 运行以后的操作

下面的表格描述了 PLC 从停止模式转换到运行模式以后的操作 (参见上面的时序图)：

阶段	描述
1	在 PLC 的停止 / 运行模式转换过程中，计时功能被触发，以便于在相当于相位 x 时基的时间周期结束时能够达到预设定的数值，从而发出第一个事件。
2	计时器的当前值 VALUE 从 0 向预先设定的值增加，每个时基脉冲会使其值加一。
3	在当前值达到了预定值以后，就会产生一个事件，计时器会返回为 0，然后重新计时。如果事件没有被掩码，相关的事件处理也会触发。如果一个具有更高或者同等优先级别的事件处理任务正在执行，那么它就会延迟处理。

事件处理中的输入 / 输出交换

常规信息

对于每一种事件处理来说，除了处理事件任务以外，还可以使用其他输入 / 输出通道。

和其他任务一样，系统会执行隐式交换，在程序之前处理 (%I)，在程序之后处理 (%Q)。

操作

下面的表格描述了被执行的交换和处理。

阶段	描述
1	如果发生了某个事件，应用程序的路径会重新安排，对引起事件的输入 / 输出通道进行相关联的处理操作。
2	系统会自动获取与事件处理相关联的输入。
3	执行事件处理。它越短越好。
4	所有与事件处理相关联的输出都会更新。

Premium/Atrium PLCs

获取输入和更新输出包括：
- 与引起事件相关联的输入通道，
- 在事件处理过程中所使用的输入和输出。

注意：这些交换可能会与以下内容相关：
- 一个通道 (比如计数模块)，
- 或者一组通道组 (离散量模块)。在这种情况下，如果数据发生改变，比如说，一个离散量模块的输出 2 和 3 发生变化，那么输出 0 到 7 的映射就会传递到该模块。

Quantum PLC

获取输入和更新输出在配置中选定。用户只能选择本地的输入 / 输出。

编程规则

在事件处理执行过程中交换的输入 (以及相关联的通道组) 会被更新 (历史值丢失，从而边沿值也会丢失)。所以您应该避免在主任务 (MAST)，快速任务 (FAST) 或者辅助 (AUXi) 任务中对这些输入的边沿进行测试。

如何编写事件处理程序

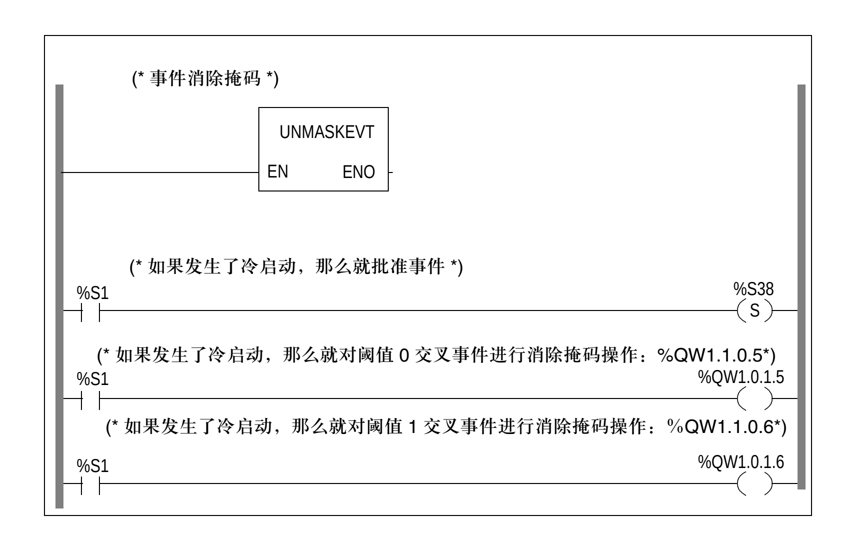
过程

下面的表格概述了编写事件处理程序的基本步骤。

步骤	操作
1	**配置阶段** (针对由输入 / 输出模块所触发的事件) 在脱机模式下，从配置编辑器内选择**事件处理** (EVT)，事件处理号，以及指定相关输入 / 输出模块通道。
2	**消除掩码阶段** 任务能够被中断的条件： ● 在系统级别激活事件处理：将 %S38 位设为 1 (缺省值)。 ● 使用 UNMASKEVT 指令对事件消除掩码 (缺省为激活状态)。 ● 在通道级别对相关事件进行消除掩码操作(针对由输入/输出模块所触发的事件)，其操作方法是将用于输入 / 输出模块中触发事件的通道，进行消除掩码操作，把其隐式语言对象设置为 1。在缺省的情况下，事件是被掩码的。 ● 在系统级别，确认事件堆栈没有饱和 (%S39 位必须是 0)。
3	创建事件程序阶段： 程序必须： ● 如果输入/输出模块能够产生多个事件，那么就要根据与输入/输出模块相关联的事件状态字来确定事件起源。 ● 执行与事件相关的映像处理。这个过程越短越好。 ● 写相关的映像输出。 **注意**：事件状态字会自动设定为 0。

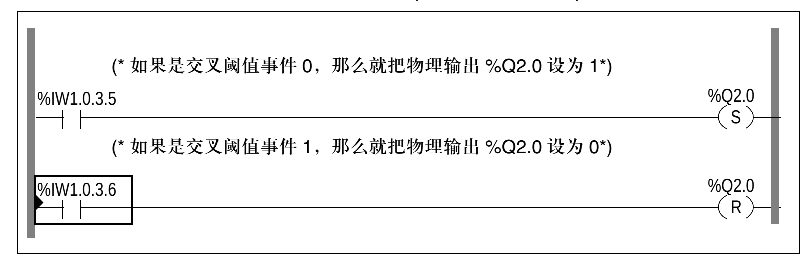
事件消除掩码图例

这个图例显示了 MAST 任务中的事件消除掩码操作。



事件处理内容图例

这个图例显示了事件处理可能含有的内容 (位测试和操作)。



应用程序内存结构

4

内容预览

本章主题 本章描述了 Premium， Atrium 和 Quantum PLC 的应用程序内存结构。

本章内容 本章包含以下内容：

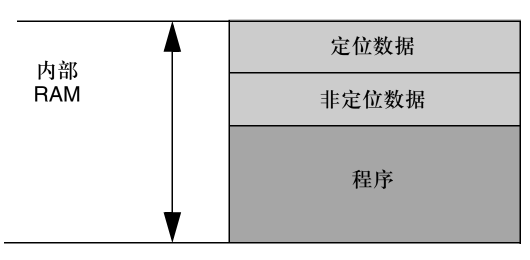
Premium 和 Atrium PLC 的内存结构

常规信息

PLC 内存支持：
- **定位型应用程序数据，**
- **非定位型应用程序数据，**
- **程序：**任务描述和可执行代码，常数字，初始值和输入 / 输出配置。

不带内存扩展卡的结构

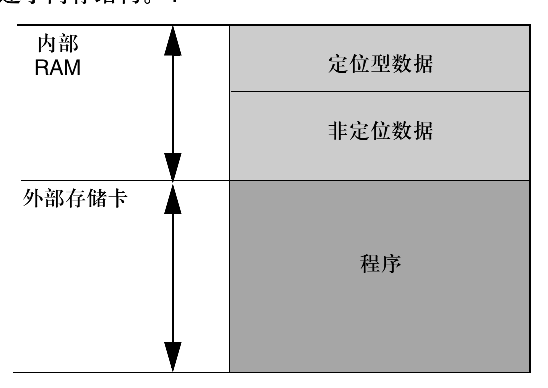
处理器模块的内部 RAM 包含了数据和程序。下面的图表描述了内存结构。 .



带有内存扩展卡的结构

处理器模块的内部 RAM 只有数据。

程序放于扩展存储卡。
下面的图表描述了内存结构。 .



内存备份

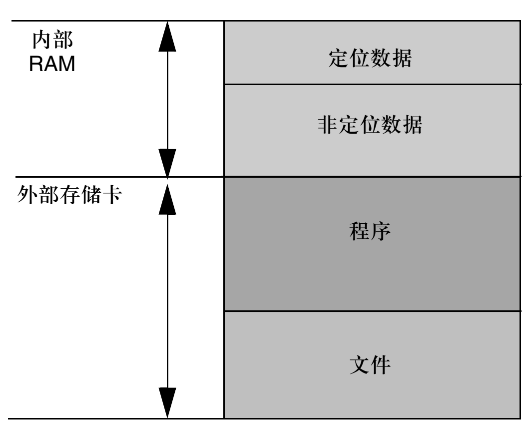
内部的 RAM 由一个 Ni-Cad 电池进行备份。
RAM 存储卡也由一个 Ni-Cad 电池进行备份。

存储卡的种类

有三种存储卡：

- **应用程序：**这种存储卡含有应用程序。它们使用 RAM 或者 Flash EPROM 技术
- **应用程序 + 文件存储：**除了程序之外，这种存储卡还包含一个区域，可以使用程序进行备份 / 恢复数据。这种存储卡使用 RAM 或者 Flash EPROM 技术
- **文件存储：**这种存储卡使用程序进行备份 / 恢复数据。它们使用 SRAM 技术

下面的图表描述了带有一个应用程序和文件存储卡的内存结构。 .



注意：在带有 2 个存储卡插槽的处理器上，较低位置的插槽预留给文件存储功能。

内存区的详细描述

用户数据

该区域包含了定位型和非定位型应用程序数据。

- 定位型数据：
  - %M， % 布尔数据和 %MW， %SW 数字数据
  - 与模块相关联的数据 (%I, %Q, %IW, %QW,%KW 等)
- 非定位型数据：
  - 布尔和数字数据 (实例)
  - EFB 和 DFB 实例

用户程序和常数

该区域包含了应用程序的可执行代码和常数。

- 可执行代码：
  - 程序代码
  - 与 EF， EFB 以及 I/O 模块管理相关的代码
  - 与 DFB 相关的代码
- 常数：
  - KW 常数字
  - 与输入 / 输出相关的常数
  - 初始数据值

该区域还包含了下载应用程序所必需的信息：图形代码，符号，等等。

其他信息

其他与应用程序配置和结构相关的信息也存储在内存中 (根据信息的类型，存储在一个数据区或者程序区)。

- 配置：其他与配置相关的数据 (硬件配置，软件配置)。
- 系统：由操作系统所使用的数据 (任务堆栈，等等)。
- 诊断：有关进程或系统的诊断，诊断缓冲器相关的信息。

Quantum PLC 的内存结构

常规信息

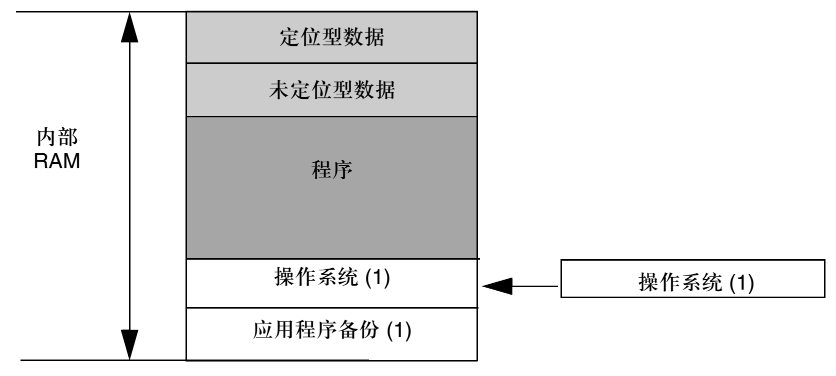
PLC 内存支持：

- **定位型应用程序数据** (State Ram)，
- **非定位型应用程序数据**，
- **程序：**任务描述和可执行代码，初始值和输入 / 输出的配置。

不带内存扩展卡的结构

处理器模块的内部 RAM 包含数据和程序。

下面的图表描述了内存结构。



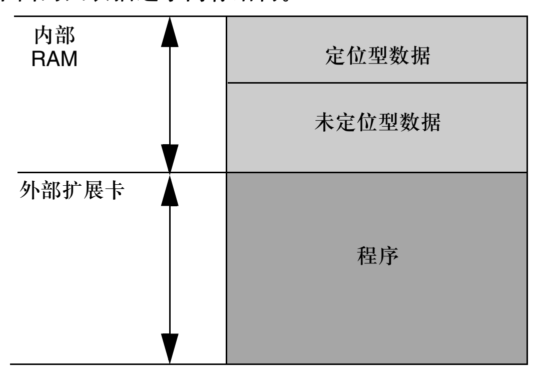
(1) 仅限于 140 CPU 31 •• /43 •• /53 •• 处理器。

带有内存扩展卡的结构

Quantum 140 CPU 6 ••• 处理器可以带有一个内存扩展卡。处理器模块的内部 RAM 只有数据。

扩展存储卡中包含程序。

下面的图表描述了内存结构。



内存备份

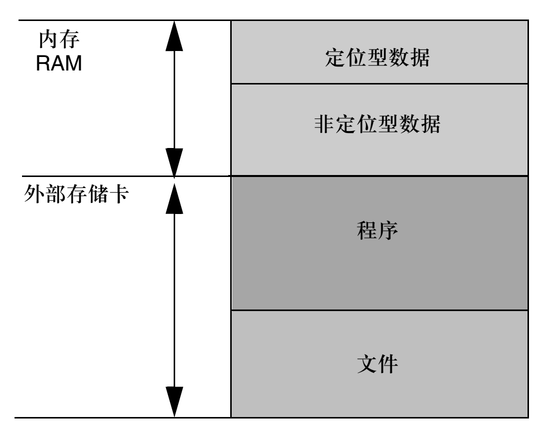
内部的 RAM 由一个 Ni-Cad 电池进行备份，该电池由处理器模块所支持。
RAM 存储卡由一个 Ni-Cad 电池进行备份。

存储卡的种类

有三种存储卡：

- **应用程序**：这种存储卡含有应用程序。它们使用 RAM 或者 Flash EPROM 技术
- **应用程序 + 文件存储**：除了程序之外，这种存储卡还包含一个区域，可以使用程序进行数据备份 / 恢复。这种存储卡使用 RAM 或者 Flash EPROM 技术
- **文件存储**：这种存储卡使用程序进行数据备份 / 恢复。它们使用了 SRAM 技术

下面的图表描述了带有一个应用程序和文件存储卡的内存结构。



> **注意**：在带有 2 个存储卡插槽的处理器上，较低位置的插槽预留给文件存储功能。

内存区的详细描述

非定位型数据

该区域包含非定位型数据：
- 布尔和数字数据
- EFB 和 DFB

定位型数据

该区域包含定位型数据 (State Ram)：

地址	对象地址	数据用途
0xxxxx	%Qr.m.c.d,%Mi	输出模块位和内部位。
1xxxxx	%Ir.m.c.d, %Ii	输入模块位。
3xxxxx	%IWr.m.c.d, %IWi	输入 / 输出模块的输入寄存器字。
4xxxxx	%QWr.m.c.d, %MWi	输入 / 输出模块的输出字和内部字。

用户程序

该区域包含应用程序的可执行代码。
- 程序代码
- EF， EFB 和 I/O 模块管理的代码
- DFB 相关的代码
- 初始变量值

该区域还包含下载应用程序所必需的信息：图形代码，符号，等等。

操作系统

在 140 CPU 31•• /41•• /51•• 处理器上，包含处理应用程序的操作系统。在启动系统电源的时候，这个操作系统从一个内部 EPROM 传送到内部 RAM 中。

应用程序备份

用户可以使用一个 1435K8 的 Flash EPROM 内存区来备份程序和变量初始值，该功能在 140 CPU 31••/41••/ 51••，处理器上可用。在 PLC 处理器电源上电的时候，存储在该区的应用程序会自动传输到内部 RAM (如果处理器前面板上的 PLC MEM 开关设定为关闭时)。

其他信息

其他与应用程序配置和结构相关的信息也存储在内存中 (根据信息的类型，存储在一个数据区或者程序区)。
- 配置：其他与配置相关的数据 (硬件配置，软件配置)。
- 系统：由操作系统所使用的数据 (任务堆栈，等等)。
- 诊断：进程或系统诊断，诊断缓冲器相关的信息。

操作模式

5

内容预览

本章主题

本章描述了 PLC 在断电和恢复过程中的操作模式，对应用程序和输入 / 输出更新的影响。

本章内容

本章包含以下内容：

断电和恢复处理

常规信息

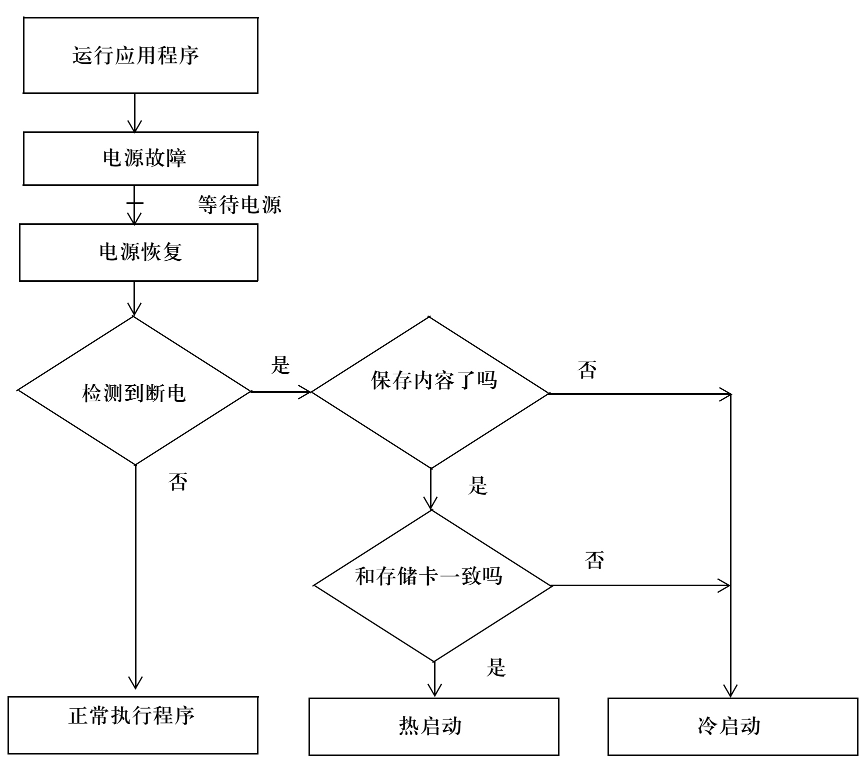
如果断电时间小于电源的过滤时间，它不会影响到程序的持续运行。如果断电时间大于电源的过滤时间，那么程序就会被中断，系统就会进行电源恢复操作。

过滤时间：

PLC	交流电流	直流电流
Quantum	10 毫秒	1 毫秒
Premium	10 毫秒	1 毫秒
Atrium	30 毫秒	-

图例

这个图例显示了由系统所检测的各种电源恢复。



操作

下面的表格描述了断电处理的各个阶段。 .

阶段	描述
1	如果发生断电，系统会存储应用程序内容以及断电时间。
2	它把所有的输出都设定为返回状态 (在配置中定义的状态)。
3	在电源恢复的过程中，系统会把保存的内容与定义的启动类型的当前内容进行比较： ● 如果应用程序内容发生了变化 (比如，系统内容或者新的应用程序丢失)，PLC 会初始化应用程序：冷启动， ● 如果应用程序的内容是一致的，PLC 会在不进行数据初始化的情况下重新启动：热启动。

除机架 0 之外的机架断电

在这个机架上的所有通道都被处理器看作是出错通道，但是其他机架则不受影响。错误的输入值在应用程序内存中不再更新，它们会在离散量输入模块中复位为 0，如果它们已经被强制赋值，那么它们则会保持该数值。

如果断电时间小于电源的过滤时间，它不会影响到程序的持续运行。

冷启动处理

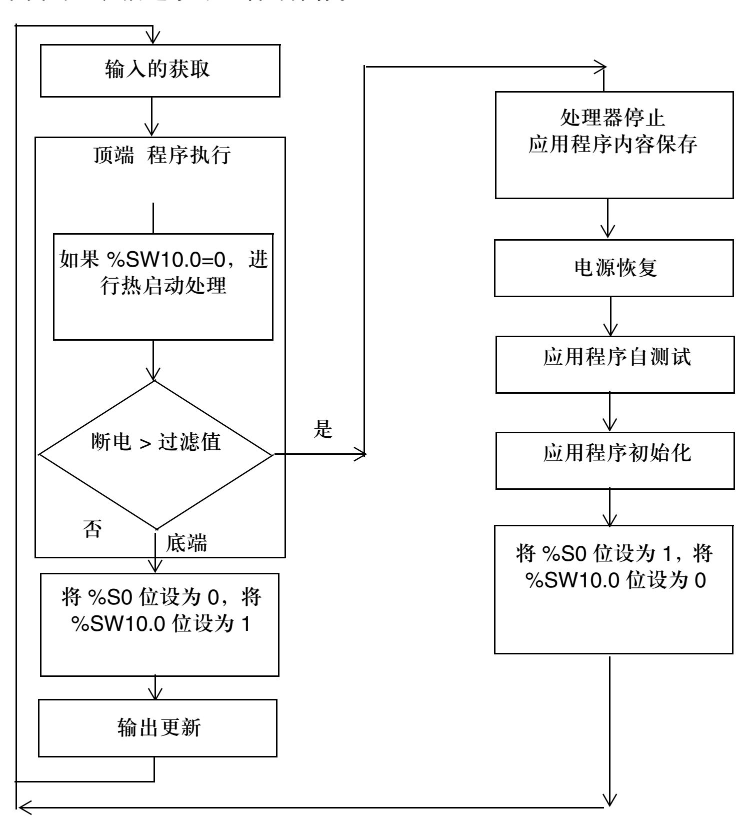
冷启动的原因

下面的表格描述了冷启动各种可能的原因。 .

原因	启动特性
加载应用程序	在停止状态下，强制冷启动
在处理器上按下了重启按钮 (Premium)	根据配置中的定义，在停止或运行模式下，强制冷启动
在发生阻塞故障以后，在处理器上按下了重启按钮 (Premium)	在停止状态下强制冷启动
对一个 PCMCIA 存储卡盖进行移动或者插入 / 拆卸了存储卡	根据配置中的定义，在停止或者运行模式下强制冷启动
从 Unity Pro 进行初始化，对 %S0 系统位强制置位	在停止或者运行模式启动 (保持断电的操作模式)，不对离散量输入 / 输出和专用模块进行初始化
断电后恢复，内容丢失	根据配置中的定义，在停止或者运行模式下强制冷启动

图例

下面的图表描述了冷重启的操作。 .



操作

下面的表格描述了在冷重启中程序执行重新启动的各个阶段。 .

阶段	描述
1	根据在配置中定义的 Automatic start in RUN 参数的状态来决定在运行或者停止模式下进行启动，或者如果它正在使用，就要根据运行 / 停止输入的状态来决定启动操作。 在周期开始的时候，程序执行会恢复。
2	系统会进行以下操作： ● 数据初始化 (位， I/O 映象，字，等等)，其中的初始值在数据编辑器中定义 (如果没有定义其他初始值，就将相关的数值设为 0)。对于 %MW 字，如果用户没有在处理器配置页面的重新启动选项中选中 “Reset”，在冷重启的时候，这些数值就会保留。 ● 根据初始数据，对基本功能块进行初始化。 ● 对在 DFB 中声明的数据进行初始化：或者将其设为 0，或者将其设为在代码中声明的初始值，或者设为在执行保存功能时所保存的数值。 ● 对系统字和位进行初始化。 ● 禁止主任务以外的任务，直到第一个主任务周期结束。 ● 把流程图定位到初始步中去。 ● 取消所有强制赋值。 ● 对消息和事件队列进行初始化。 ● 把配置参数发送给所有离散量输入 / 输出模块以及专用模块。
3	在第一个重新启动周期，系统会进行以下操作： ● 重新启动主任务，把 %S0 (冷重启) 和 %S13 (运行模式下的第一个周期) 位设为 1，把 %SW10 字 (在第一个任务周期中进行冷启动检测) 设为 0， ● 在主任务的第一个周期结束时把 %S0 和 %S13 位重置为 0，把 %SW10 字的每一位都设为 1， ● 在主任务的第一个周期结束时激活 fast 任务和事件处理。

通过一个程序处理冷启动

建议您对 %SW10.0 位进行测试，以便于检测冷启动，并开始与该冷启动相关的处理过程。

> **注意：**如果已经选择了 **Automatic start in RUN** 参数，就可以测试 %S0 位。如果没有选择该参数， PLC 会从停止模式开始运行，接下来，在重新启动之后， %S0 位会在第一个周期置 1，但是因为没有被执行，所以它对于程序来说是不可见的。

Premium 和 Atrium 的输出变化

一旦检测到了断电，输出就会置到返回置，根据在配置中所作的选择：
- 系统或者为它们分配返回值，
- 或者保留当前值。

在电源恢复以后，输出会保持为零，直到被任务所更新。

Quantum 的输出变化

一旦检测到了断电，
- 本地输出就会设为零，
- 远程或者分布式扩展机架的输出会被设置到返回置。

在电源恢复以后，输出会保持为零，直到被任务所更新。

Quantum 140 CPU 31••/41••/51•• 的情况

这些处理器拥有一个 1435K8 的 Flash EPROM 内存，可以用来保存程序和变量的初始值。
在电源恢复以后，您可以通过处理器前面板上的 PLC MEM 开关来选择所需的操作模式。
- **关闭位置**：在 PLC 处理器电源上电的时候，该区域所包含的应用程序会自动传输到内部的 RAM：应用程序冷启动。

开启位置：该区域所包含的应用程序不会传输到内部的 RAM：应用程序热启动。

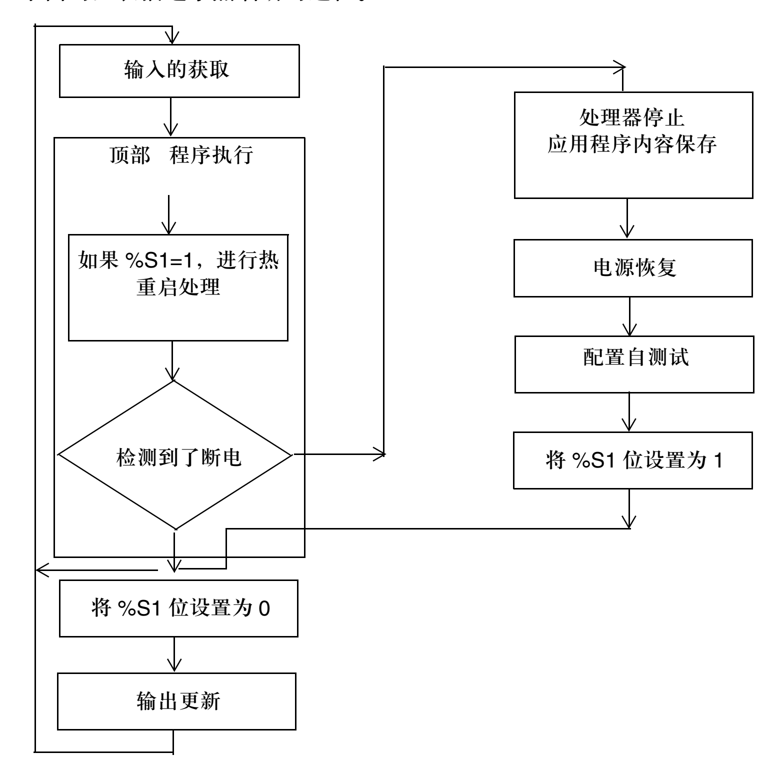
处理热启动

热启动的原因

热启动可能有以下原因：
- 电源恢复，内容没有丢失，
- 程序将系统位 %S1 设置为 1，
- 通过来自终端的 Unity Pro 进行热启动，
- 按下机架 0 (在 Premium PLC 上) 电源模块上的重启动按钮。

图例

下面的图表描述了热启动的过程。



操作

下面的表格描述了在热启动中程序执行重新启动的各个阶段。 .

阶段	描述
1	程序执行从发生断电的元素处恢复启动操作，不更新输出。
2	在重新启动周期结束的时候，系统进行以下操作： ● 消息和事件队列的初始化， ● 把配置参数发送给所有离散量输入 / 输出模块以及专用模块， ● 重新激活快速任务和事件处理 (直到主任务周期结束)。
3	系统会执行一个重新启动周期，在该周期中： ● 对所有输入模块进行重新确认， ● 重新启动主任务，将 %S1(热重启) 位设置为 1， ● 在主任务第一个周期结束的时候，将 %S1 位置为 0，

通过程序处理热启动

在发生了热动启的情况下，如果您想对应用程序进行特殊处理，就必须在主任务开始的时候把相应程序的测试条件定为：%S1 = 1。

Premium 和 Atrium 的输出变化

一旦检测到了断电，输出就会被设置到返回，根据在配置中所作的选择：
- 系统或者为它们分配返回值，
- 或者保留当前值。

在电源恢复以后，输出会保持为零，直到被任务所更新。

Quantum 的输出变化

一旦检测到了断电，
- 本地输出会被设置为零，
- 远程或者分布式扩展机架的输出会被设置到返回。

在电源恢复以后，输出会保持为零，直到被任务所更新。

系统对象

6

内容预览

本章主题

本章描述了 Unity Pro 语言的系统位和字。

注意：与每一个位对象或者系统字相关的符号都会在这些对象的说明表格中给出，它们不是软件中的标准，但是可以通过数据编辑器输入。
这些符号是按照顺序排列的，这样能够确保在不同的应用程序中保持一致。

本章内容

本章包含以下各节：

6.1 系统位

描述

本节主题

本节描述了系统字。

本节内容

本节包含以下主题：

系统位介绍

常规信息

Premium， Atrium 和 Quantum PLC 使用的 %Si 系统位用来显示 PLC 状态，它们也可以用来控制 PLC 的操作。

可以在用户程序中测试这些系统位，以便于对需要用到一套处理程序的功能开发进行检测。

程序在使用某些系统位的时候，必须把它们置为初始值，或者正常状态。但是，那些被系统置为初始值或者正常状态的系统位，则不能用程序或者终端复位。

系统位 %S0 到 %S7 描述

详细的描述

系统位 %S0 到 %S7 描述：

位 符号	功能	描述	初始状态	Quantum	Premium Atrium
%S0 COLDSTART	冷启动	通常情况下为 0，可以通过以下方式置为 1： ● 电源恢复，数据丢失 (电池故障)， ● 用户程序， ● 终端， ● 存储卡变化， ● 加载程序。 不管是在运行模式还是停止模式，在 PLC 第一个完整的恢复周期内，该系统位都会被设定为 1。它在下一个周期之前被系统重置为零。	1 (1 周期)	有	有
%S1 WARMSTART	热启动	通常情况下为 0，可以通过以下方式设为 1： ● 电源恢复，数据被保存， ● 用户程序， ● 终端， ● 存储卡更改操作。 该系统位会在第一个完整的周期结束的时候，在输出更新之前被系统重置为 0。	0	有	有
%S4 TB10MS	时基 10 毫秒	一个内部计时器会控制该系统位状态的更改。它与 PLC 周期是不同步的。 图形： 5 毫秒 5 毫秒	-	有	有
%S5 TB100MS	时基 100 毫秒	同 %S4	-	有	有
%S6 TB1SEC	时基 1 秒	同 %S4	-	有	有
%S7 TB1MIN	时基 1 分钟	同 %S4	-	有	有

系统位 %S9 到 %S13 描述

详细的描述

系统位 %S9 到 %S13 描述：

位 符号	功能	描述	初始 状态	Quantum	Premium Atrium
%S9 OUTDIS	在所有总线上的输出设置到返回置	正常情况下为 0，由程序或者终端置为 1： ● 设置为 1：根据所选的配置 (X 总线，Fipio，AS-i，等等)，将该位置为 0，或者保留当前值配置。 ● 设置为 0：输出正常更新。 **注意：**系统位直接作用于物理输出，不作用于系统的映像位。	0	无	有
%S10 IOERR	输入 / 输出错误	正常情况下为 1，当检测到一个机架模块或者 Fipio 设备发生 I/O 故障的时候，该系统位置为 0 (比如 不兼容的配置，交换错误，硬件错误，等等)。一旦错误消失，系统会马上把它复位为 1。	1	有	有
%S11 WDG	监视时钟溢出	正常情况下为 0，一旦任务执行时间大于在任务属性中所声明的最大执行时间 (也就是监视时钟)，它会马上置为 1。	0	有	有
%S12 PLCRUNNING	PLC 处于运行模式	当 PLC 处于运行模式时，该系统位会被系统设为 1 。 只要 PLC 不再处于运行模式 (停止模式，初始模式，等等)，系统就会马上把它置为 0。	0	有	有
%S13 1RSTSCANRUN	在转换到运行模式以后的第一个周期导通	通常情况下被设定为 0，在 PLC 设置为运行模式以后，在第一个主任务期间会被系统设置为 1 。	-	有	有

系统位 %S15 到 %S20 描述

详细的描述

系统位 %S15 到 %S20 描述：

位 符号	功能	描述	初始 状态	Quantum	Premium Atrium
%S15 STRINGERROR	字符串错误	通常情况下被设定为 0，如果字符串传输的目标区尺寸不够大，无法接收该字符串，那么该系统位会置为 1。 如果 %S78 位已经被设置为 1，那么该应用程序会停留在错误状态。 该系统位必须用应用程序复位为 0。	0	有	有
%S16 IOERRTSK	任务输入 / 输出错误	通常情况下置为 1，如果在任务中配置的一个机架 I/O 模块或者 Fipio 设备出错，它会被系统设定为 0。该系统位必须被用户复位为 1。	1	有	有
%S17 CARRY	循环移位输出	通常情况下为 0。 在一个循环移位操作中，该操作位输出为 1。	0	有	有

位 符号	功能	描述	初始状态	Quantum	Premium Atrium
%S18 OVERFLOW	溢出或者算术错误	正常情况下置为 0，如果因以下情况而发生容量溢出，它会被置为 1： ● 对单字长来说，某个结果 > + 32 767 或者 < - 32 768, ● 对无符号整数来说，结果 > + 65 535, ● 对双字长来说，某个结果 > + 2 147 483 647 或者 -< 2 147 483 648, ● 对双字长或者无符号整数来说，结果 > +4 294 967 296, ● 超出范围的实数值， ● 除数为 0， ● 负数的平房根， ● 在一个步进控制器上对一个不存在的步强制操作。 ● 向已满的寄存器内加入数据，或者清空已空的寄存器。 在每次操作以后，用户程序都必须在可能发生溢出的地方测试该系统位，如果的确发生了溢出，那么用户就要把该系统位重置为 0。 在 %S18 位置 1 以后，如果 %S78 位已经置为 1，那么应用程序就会在错误的状态停止。	0	有	有
%S19 OVERRUN	任务周期溢出（周期性扫描）	通常情况下置为 0，如果发生了时间周期溢出，系统就会把它置为 1（也就是说任务执行时间大于用户在配置中定义的周期或者在与任务相关的 %SW 字中所编写的周期）。用户必须把这个系统位复位为 0。每一个任务都管理自己的 %S19 位。	0	有	有

位 符号	功能	描述	初始 状态	Quantum	Premium Atrium
%S20 INDEXOVF	索引溢出	正常情况下置为 0，当索引对象的地址小于 0 或者超过在配置中所声明的对象地址时，它会被设为 1，就相当于索引为 0 的情况。 在每次操作以后，用户程序都必须在可能发生溢出的地方测试该系统位，如果的确发生了溢出，那么用户就要把该系统位复位为 0。 在 %S20 位置 1 以后，如果 %S78 位已经置为 1，那么应用程序就会在错误的状态停止。	0	有	有

系统位 %S30 到 %S59 描述

详细的描述

系统位 %S30 到 %S59 描述：

位 符号	功能	描述	初始状态	Quantum	Premium Atrium
%S30 MASTACT	激活 / 禁止主任务	正常情况下置为 1，用户如果将该位置为 0，那么就会禁止主任务。 在每一个 MAST 任务周期结束的时候，系统都会考虑到该系统位的情况。	1	有	有
%S31 FASTACT	激活 / 禁止快速任务	同 %S30，用于 FAST 任务。	0	有	有
%S32 AUX0ACT	激活 / 禁止辅助任务 0	同 %S30，用于 AUX0 任务。	0	有	有
%S33 AUX1ACT	激活 / 禁止辅助任务 1	同 %S30，用于 AUX1 任务。	0	有	有
%S34 AUX2ACT	激活 / 禁止辅助任务 2	同 %S30，用于 AUX2 任务。	0	有	有
%S35 AUX3ACT	激活 / 禁止辅助任务 3	同 %S30，用于 AUX3 任务。	0	有	有
%S38 ACTIVEVT	激活 / 禁止事件	正常情况下置为 1，用户如果将该置为 0，那么就会禁止事件。	1	有	有
%S39 EVTOVR	事件处理饱和	如果系统将该位置为 1，表示已经队列饱和，无法处理一个或多个事件。用户必须将该位复位为 0。	0	有	有

位 符号	功能	描述	初始状态	Quantum	Premium Atrium
%S40 RACK0ERR	机架 0 输入 / 输出错误	%S40 位被分配给机架 0。 该位正常情况下置为 1，如果机架 I/O 发生了错误，那么它就会被设为 0。 在这种情况下： ● %S10 位置为 0， ● I/O 处理器发光二极管点亮， ● %Ir.m.c.Err 模块置为 1。 在错误消失以后，该位会复位为 1。	1	无	有
%S41 RACK1ERR	机架 1 输入 / 输出错误	同 %S40，用于机架 1。	1	无	有
%S42 RACK2ERR	机架 2 输入 / 输出错误	同 %S40，用于机架 2。	1	无	有
%S43 RACK3ERR	机架 3 输入 / 输出错误	同 %S40，用于机架 3。	1	无	有
%S44 RACK4ERR	机架 4 输入 / 输出错误	同 %S40，用于机架 4。	1	无	有
%S45 RACK5ERR	机架 5 输入 / 输出错误	同 %S40，用于机架 5。	1	无	有
%S46 RACK6ERR	机架 6 输入 / 输出错误	同 %S40，用于机架 6。	1	无	有
%S47 RACK7ERR	机架 7 输入 / 输出错误	同 %S40，用于机架 7。	1	无	有
%S50 RTCWRITE	通过字 %SW50 到 %SW53 更新时间和日期	该位通常情况下置为 0，可以通过程序或者终端将其设为 1。 ● 设为 0：用 PLC 实时时钟所提供的日期和时间来更新系统字 %SW50 到 %SW53。 ● 设为 1：不再更新系统字 %SW50 到 %SW53，从而使用户能够修改它们。 ● 该位如果从 1 变到 0，系统用输入的系统字 %SW50 到 %SW53 对实时时钟进行更新。	0	有	有

位 符号	功能	描述	初始 状态	Quantum	Premium Atrium
%S51 RTCERR	实时时钟时间丢失	该位由系统管理，如果它置为 1，就表示实时时钟丢失，或者它的系统字 (%SW50 到 %SW53) 没有意义。在这种情况下，用户必须把时钟复位到正确时间。	-	有	有
%S59 RTCTUNING	通过 %SW59 字对时间和日期进行递增的调整	该位正常情况下置为 0，可以通过程序或者终端将其置为 1： ● 设为 0：系统不管理系统字 %SW59， ● 设为 1：系统管理 %SW59 字，通过上升沿，可以调整日期和当前时间 (通过递增的方式)。	0	有	有

系统位 %S60 到 %S78 描述

详细的描述

系统位 %S60 到 %S78 描述：

位 符号	功能	描述	初始状态	Quantum	Premium Atrium
%S60 BACKUPCHVOV	主动转换命令	在使用冗余体系结构的场合下，该系统位控制用来主动转换。 该系统位可以通过用户或者应用程序复位为 0。	0	无	有
%S67 PCMCIABAT0	应用程序存储卡电池的状态	该系统位用来控制 RAM 内存卡电池 (插在顶部的插槽内) 的状态： ● 置为 0：有电池，工作正常 ● 置为 1：电池丢失，或工作不正常。 **注意：**在以下 Quantum 中不具备该功能：140CPU 31 •• /43 •• and 53 ••	-	有	有
%S68 PLCBAT	处理器电池的状态	该系统位用来检查 RAM 中用来保存数据和程序的电池的操作状态。 ● 置为 0：有电池，工作正常 ● 置为 1：电池丢失，或工作不正常。	-	有	有
%S75 PCMCIABAT1	数据存储存储卡电池的状态	该系统位用来控制数据存储内存卡电池 (插在较低的插槽内) 的状态： ● 设为 0：有可用的电池， ● 设为 1：电池丢失，或者不可用。 **注意：**在以下 PLC 中不具备该功能： Quantum 140CPU 31 •• /43 •• and 53 •• ， Premium TSX P 57 1 •• /2 •• /3 •• /4 •• 以及 Atrium	-	有	有

位 符号	功能	描述	初始状态	Quantum	Premium Atrium
%S76 DIAGBUFFCONF	配置诊断缓冲器	在诊断选项被配置以后，该系统位为 1 时，系统会保留一个诊断缓冲器，用来存储由诊断 DFB 所发现的错误。 该系统位可以由用户或者应用程序复位为 0。	0	有	有
%S77 DIAGBUFFFFULL	诊断缓冲器充满	如果从诊断功能块接收错误把缓冲器满了，那么该位就会被系统置为 1。 该系统位可以由用户或应用程序复位为 0。	0	有	有
%S78 HALTIFERROR	在发生错误的时候停止 PLC	该位正常情况下置为 0，它可以由用户设置为 1，从而在应用程序发生错误的时候：如 %S15， %S18， %20 置 1 时，使 PLC 停止工作。	0	有	有

系统位 %S80 到 %S94 描述

详细的描述 系统位 %S80 到 %S94 描述：

位 符号	功能	描述	初始状态	Quantum	Premium Atrium
%S80 RSTMSGCNT	对消息计数器进行复位	该位正通常情况下置为 0，用户可以把它设定为 1，可以对消息计数器 %SW80 到 %SW86 进行复位。	0	有	有
%S90 COMRFSH	更新公共字	该位通常情况下被设定为 0，在从另外一个网络工作站接收到公共字以后，它会被设定为 1。 该位可以被程序或者终端设定为 0，以便于检查公共字交换周期。	0	无	有
%S91 LCKASYNREQ	锁定异步请求	如果该位置为 1，在监视任务处理时，保证异步通信请求会完整地执行，不会被其他 MAST 或者 FAST 任务中断，从而能够确保数据在读写过程中保持连贯。 **请注意：**监视任务的请求服务器通过 7 号门 (X-Way) 进行寻址。	0	无	有
%S92 EXCHGTIME	通信功能测量模式	该位正常情况下置为 0，用户可以把它设定为 1，以便于将通信功能设置为性能测量模式。 通信功能的往返交换时间会以 10 毫秒为单位显示出来。 **注意：**通信功能会以 10 秒钟的标准使用时间来执行。	0	无	有
%S94 SAVECURRVAL	保存调整值	该位通常情况下为 0，用户可以把它设为 1，以便于将带有 “保存” 属性的声明变量 (比如：DFB 变量) 初始值用当前值代替。在代替操作完毕以后，系统会将 %S94 位复位为 0。 注意：在使用这个系统位的时候请务必小心：不要将其永久性地设为 1，该位仅用于主任务。	0	有	有

系统位 %S100 到 %S122 描述

详细的描述

系统位 %S100 到 %S122 描述：

位 符号	功能	描述	初始状态	Quantum	Premium Atrium
%S100 `PROTTERINL`	终端口上的协议	根据控制台上的 INL/DPT 分路的状态，该位会被系统设置为 0 或 1 ● 如果该分路缺失 (%S100=0)，那么系统就会使用 Uni-Telway 主协议， ● 如果该分路存在 (%S100=1)，那么系统就会使用在应用程序配置中所指定的协议。	-	无	有
%S118 `REMIOERR`	常规 Fipio I/O 错误	该位正常情况下为 1，如果与 Fipio 连接的某个设备上发生了错误，系统会把该位设为 0。在错误消失以后，系统会把它复位为 1。	-	有	有
%S119 `LOCIOERR`	常规机架 I/O 错误	该位正常情况下置为 1，如果在某个机架上的一个 I/O 发生了错误，系统会把该位置为 0。在错误消失以后，系统会把它复位为 1。	-	有	有
%S120 `DIOERRPLC`	DIO 总线错误 (CPU)	该位通常情况下被设置为 1，如果与配置给 CPU 的 Modbus Plus 链接所管理的 DIO 总线相连的设备发生了错误，系统会把该位置为 0。	-	有	无
%S121 `DIOERRŒ?M1`	DIO 总线错误 (1 号无 M)	该位通常情况下被设置为 1，如果与被第一个 140 无 M 2 •• 模块所管理的 DIO 总线相连的设备发生了错误，系统会把该位置为 0。	-	有	无
%S122 `DIOERRŒ?M2`	DIO 总线 (2 号无 M)	该位通常情况下被设置为 1，如果与被第二个 140 无 M 2 •• 模块所管理的 DIO 总线相连的设备发生了错误，系统会把该位置为 0。	-	有	无

6.2 系统字

描述

本节主题

本章描述了 Atrium， Premium 和 Quantum 的系统字。

本节内容

本节包含以下主题：:

系统字 %SW0 到 %SW11 描述

详细的描述

系统字 %SW0 到 %SW11 描述：

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW0 MASTPERIOD	主任务扫描周期	可使用用户程序或者终端修改主任务周期。 周期以毫秒为单位 (1...255 ms) 计算。 在循环操作中， %SW0=0。 在冷启动时：它会使用在配置中所定义的数值。	0	有	有
%SW1 FASTPERIOD	快速任务扫描周期	可使用用户程序或者终端修改快速任务周期。 周期以毫秒为单位 (1...255 ms) 计算。 在冷启动时：它会使用在配置中所定义的数值。	0	有	有
%SW2 AUX0PERIOD **%SW3** AUX1PERIOD **%SW4** AUX2PERIOD **%SW5** AUX3PERIOD	辅助任务扫描周期	可使用用户程序或者终端修改快速任务周期。 周期以 10 ms 为单位 (10 ms...2.55 s) 计算。 (1) 仅用于 140 CPU 6 •• 和 TSX 57 5 •• PLC。	0	有 (1)	有 (1)
%SW8 TSKINHIBIN	输入获取任务控制	正常情况下置为 0，可以用程序或者终端置为 1 或 0。它禁止每一个任务的输入获取阶段： • %SW8.0 = 1 禁止与 MAST 任务相关的输入获取。 • %SW8.1 = 1 禁止与 FAST 任务相关的输入获取。 • %SW8.2 到 5 = 1 禁止与 AUX 0...3 任务相关的输入获取 (1) 注意：在 Quantum 上，通过 DIO 总线实现的分布式输入 / 输出不通过 %SW8 字进行分配。	0	有 (1)	有

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW9 TSKINHIBOUT	任务输出更新的控制	正常情况下置为 0，可以被程序或者终端设置为 1 和 0。 它禁止每一个任务的输出更新阶段： ● %SW9.0 = 1 分配给 MAST 任务；系统不再管理与该任务相关的输出。 ● %SW9.1 = 1 分配给 FAST 任务；系统不再管理与该任务相关的输出。 ● %SW9.2 到 5 = 1 分配给 AUX 0...3 任务；系统不再管理与该任务相关的输出。 (2) **注意**：在 Quantum 上，通过 DIO 总线实现的分布式输入 / 输出不通过 %SW9 字进行分配。	0	有 (2)	有
%SW10 TSKINIT	冷启动以后的第一个周期	如果当前任务位的数值为 0，就表示任务在冷启动以后正在执行第一个周期。 ● %SW10.0：分配给 MAST 任务。 ● %SW10.1：分配给 FAST 任务。 ● %SW10.2 到 5：分配给 AUX 0...3 任务。	0	有	有
%SW11 WDGVALUE	监视时钟持续时间	读出监视时钟的持续时间。持续时间以毫秒为单位 (10...1500 毫秒)。该系统字不能被修改。	-	有	有



警报

关于 %SW9 系统字：

在 Premium 上：

连接到 X 总线的模块的输出会自动转换到配置的模式 (返回或保持)。连接到 Fipio 总线的设备的输出会处于保持状态。

在 Quantum 上：

本地机架模块和远程 (RIO) 机架模块的输出会处于保持状态。

如果未能遵守此处的警告，可能会造成伤害或者设备损坏。

系统字 %SW12 到 %SW18 描述

详细的描述　　系统字 %SW12 到 %SW18 描述：

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW12 UTWPORTADDR	Uni-Telway 终端端口 地址	在配置中所定义的终端口 Uni_Telway 的地址 (在从模式时)，在冷启动的时候会被加载到这个字中去。 **注意：**系统不会考虑对这个字所做的修改。	-	无	有
%SW13 XWAYNETWADDR	站的主地址	给出网络的以下内容 (Fipway 或者 Ethway)： ● 站号 (最低字节) 从 0 到 127， ● 网络号 (最高字节) 从 0 到 63， (在 PCMCIA 卡上的拨码盘的号)。	254 (16#00FE)	无	有
%SW14 OSCOMMVERS	PLC 处理器 的商业版本	该系统字包含了 PLC 处理器的商业版本。 例子：16#0135 版本：01 发行号：35	-	有	有
%SW15 OSCOMMPATCH	PLC 处理器 补丁版本	该系统字包含了 PLC 处理器商业版本的补丁号。 它的编码是该系统字的最低字节。 代码：0 = 没有补丁， 1 = A, 2 = B... 例子：16#0003 表示补丁 C。	-	有	有
%SW16 OSINTVERS	PLC 处理器 的固件版本	该系统字包含了 PLC 处理器固件的版本。 例子：16#0143 版本：01 发行号：43	-	有	有

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW17 FLOATSTAT	浮点运算状态错误	如果检测到浮点算术运算出错， %S18 位会置为 1， %SW17 错误状态会根据以下代码进行更新： ● %SW17.0 = 无效运算 / 运算结果不是一个数值 ● %SW17.1 = 不标准的操作数 / 结果可以接受 ● %SW17.2 = 除数为 0/ 结果无穷大 ● %SW17.3 = 溢出 / 结果无穷大 ● %SW17.4 = 下溢 / 结果为 0 ● %SW17.5 到 15 = 未使用 在冷启动的时候，该字被系统复位为 0，它也可以出于使用的目的而被程序复位为 0。	0	有	有
%SD18 100MSCOUNTER	绝对时间计时器	这个双字用来计算持续时间。 每十分之一秒，系统会把它累加一次 (即使当 PLC 于停止状态也如此，如果 PLC 断电，它就不再累加)。用户程序或者终端可以对它进行读写操作。	0	有	有

系统字 %SW30 到 %SW47 描述

详细的描述 系统字 %SW30 到 %SW35 描述：

字 符号	功能	描述	初始状态	Quantum	Premium
%SW30 MASTCURRTIME	主任务执行时间	说明上一个主任务周期的执行时间 (用毫秒表示)。	-	有	有
%SW31 MASTMAXTIME	最大主任务执行时间	说明从上次冷启动以后最长的主任务执行时间 (用毫秒表示)。	-	有	有
%SW32 MASTMINTIME	最小主任务执行时间	说明从上次冷启动以后最短的主任务执行时间 (用毫秒表示)。	-	有	有
%SW33 FASTCURRTIME	快速任务执行时间	说明上一个快速任务周期的执行时间 (用毫秒表示)	-	有	有
%SW34 FASTMAXTIME	最大快速任务执行时间	说明从上次冷启动以后最长的快速任务执行时间 (用毫秒表示)。	-	有	有
%SW35 FASTMINTIME	最小快速任务执行时间	说明从上次冷启动以后最短的快速任务执行时间 (用毫秒表示)。	-	有	有

注意：执行时间是在一个扫描周期开始 (输入获取) 和结束 (输出更新) 之间的实耗时间。这个时间包括事件任务，快速任务处理，以及控制台请求处理。

系统字 %SW36 到 %SW47 描述：

字 符号	功能	描述	初始 状态	Quantum	Premium
%SW36 AUX0CURRTIME **%SW39** AUX1CURRTIME **%SW42** AUX2CURRTIME **%SW45** AUX3CURRTIME	辅助任务执行时间	说明 AUX 0...3 任务上一个周期的执行时间 (用毫秒表示)。 (1) 仅限于 140 CPU 6 •• 和 TSX P57 5 •• PLC。	-	有 (1)	有 (1)
%SW37 AUX0MAXTIME **%SW40** AUX1MAXTIME **%SW43** AUX2MAXTIME **%SW46** AUX3MAXTIME	最大辅助任务执行时间	说明从上一次冷启动以后 AUX 0...3 任务的最长任务执行时间 (用毫秒表示)。 (1) 仅限于 140 CPU 6 •• 和 TSX P57 5 •• PLC。	-	有 (1)	有 (1)
%SW38 AUX0MINTIME **%SW41** AUX1MINTIME **%SW44** AUX2MINTIME **%SW47** AUX3MINTIME	最小辅助任务执行时间	说明从上一次冷启动以后 AUX 0...3 任务的最短任务执行时间 (用毫秒表示)。 (1) 仅限于 140 CPU 6 •• 和 TSX P57 5 •• PLC。	-	有 (1)	有 (1)

系统字 %SW48 到 %SW59 描述

详细的描述

系统字 %SW48 到 %SW59 描述：

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW48 IOEVTNB	事件的数量	说明上次冷启动以后所处理的事件的数量。程序或者终端可以对它进行写操作。	0	有	有
%SW49 DAYOFWEEK **%SW50** SEC **%SW51** HOURMIN **%SW52** MONTHDAY **%SW53** YEAR	实时时钟功能	包含日期和当前时间的系统字 (BCD 格式)： • %SW49：星期几 • 1 = 星期一 • 2 = 星期二 • 3 = 星期三 • 4 = 星期四 • 5 = 星期五 • 6 = 星期六 • 7 = 星期日 • %SW50：秒 (16#SS00) • %SW51：小时和分钟 (16#HHMM) • %SW52：月和日 (16#MMDD) • %SW53：年 (16#YYYY) 当 %S50 被设置为 0 的时候，这些字由系统来管理。 当 %S50 被设置为 1 的时候，用户程序或者终端可以对它们进行写操作。	-	有	有

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW54 STOPSEC **%SW55** STOPHM **%SW56** STOPMD **%SW57** STOPYEAR **%SW58** STOPDAY	上一次停止时的实时时钟功能	系统字包含了上一次电源故障或者 PLC 停止的日期和时间 (BCD 格式)： ● %SW54: 秒 (00SS) ● %SW55: 小时和分钟 (HHMM) ● %SW56: 月和日 (MMDD) ● %SW57: 年 (YYYY)。 ● %SW58最高字节包含了星期几的信息(1表示星期一，以此类推，直到 7 表示星期日)。 ● %SW58 最低字节包含了上一次停止的代码。 ● 1 = 通过终端或者专用的输入位从运行模式转换到停止模式 ● 2 = 因软件错误(PLC任务或者SFC溢出)而停止 ● 4 = 断电或者存储卡操作 ● 5 = 因硬件错误而停止 ● 6 = 因暂停指令而停止	-	有	有
%SW59 ADJDATETIME	调整当前日期	包含两个 8 位字段，用来调整当前日期。 相关操作总是在位的上升沿来完成的。 这个系统字通过 %S59 位激活。 图例： + - 位 0 8 星期几 (1) 1 9 秒 2 10 分 3 11 小时 4 12 天 5 13 月 6 14 年 7 15 世纪	0	有	有

详细的描述

系统字 %SW60 到 %SW69 描述：

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW60 WSBPLCDIAG0	备份 PLC 诊断	专门用于本地 PLC 备份的诊断 (Warm Standby Premium) %SW60 字中各位的含义： ● %SW60.0=1 PLC 处于 “正常” 状态 ● %SW60.1=1 PLC 处于 “备份” 状态 ● %SW60.3=0 PLC 处于 “正常” 状态的情况下， Fipio 的 I/O 发生错误；它代表 %S118 位的值 ● %SW60.4=0 机架的I/O发生错误；它代表了 %S119 位的值 ● %SW60.5=1 至少有一个 TSX ETY 210 模块在自测试的过程中检测到了错误 ● %SW60.7=1 Fipio 网络发生严重错误，比如短路或者终端块断开 ● %SW60.8=1 用于 PLC 间链接的 TSX ETY 110 模块发生错误 ● %SW60.9=1 PLC 间的通信发生错误 – 不能恢复双重 PLC 诊断 ● %SW60.10 是一个保留位 ● %SW60.11=1 备份 PLC 不适合充当 “正常的” PLC。该信息只在一个正常的 PLC 上产生，对备份 PLC 没有意义。 ● %SW60.12=0 有效的 PLC 是 A 工作站 ● %SW60.12=1 有效的 PLC 是 B 工作站 ● %SW60.13=1 PLC 运行模式 ● %SW60.14=1 PLC 停止模式 ● %SW60.15=1 PLC 暂停模式	0	无	有

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW60 WSBPLCDIAG0	Quantum 热备用命令寄存器	%SW60 字中各位的含义： ● %SW60.5 = 0 在一个主要交换中允许在端口 3 进行地址转换。 ● %SW60.6=0 在一个主要交换中允许在端口 2 进行地址转换。 ● %SW60.7=0在一个主要交换中允许在端口1 进行地址转换。 ● %SW60.10=1 从备用设备到主设备的应用程序传输请求。 ● %SW60.11=0 只有在应用程序停止以后，才允许更新固件。 ● %SW60.12=0 如果备用应用程序使用一个与主要应用程序不同的逻辑，那么就强制进入脱机备用模式。 ● %SW60.13 　● =0 把控制器 B 转换到脱机模式。 　● =1 把控制器 B 转换到运行模式。 ● %SW60.14 　● =0 把控制器 A 转换到脱机模式。 　● =1 把控制器 A 转换到运行模式。 ● %SW60.15=0 使显示器(键盘)中输入的命令成为无效。	0	无	有

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW61 WSBPLCDIAG0	备份 PLC 诊断	%SW61 字中各位的含义： ● %SW61.0=1 表示通过 Ethway PLC 间链接所进行的数据库交换出现了问题。只有处于运行模式的正常 PLC 能够产生该信息。 ● %SW61.1=1 表示在某一个第三方设备的 TSX ETY 210 TCP/IP 客户端模块之间的通信出现了问题。只有处于运行模式的正常 PLC 能够产生该信息。在该位变为 1 的情况下，如果备份 PLC 适合充当正常 PLC，那么就会进行转换。 ● %SW61.4=1 表示第一次正确的数据库交换。 ● %SW61.5=1 表示处理器被备份功能所中止。在 %MWp.14.2 字中会给出诊断 ● %SW61.7=0 表示备份功能的配置或者操作出现了问题。在 %MWp.14.2 字中会给出诊断 ● %SW61.7=1 表示已经正确配置了备份功能 ● %SW61.1，3，8 到 15 是保留位 p：表示处理器插槽	0	无	有

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW61 WSBPLCDIAG0	Quantum 状态寄存器	%SW61 字中各位的含义： ● %SW61.0=0 非定位型变量被传输。 ● %SW61.1=0 协处理器正在以正确的方式操作。 ● %SW61.10 ● =0 PLC 作为 A 设备使用。 ● =1 PLC 作为 B 设备使用。 ● %SW61.11=0 PLC 上的应用程序是一致的。 ● %SW61.12和%SW61.13 来自其他PLC的操作模式位 ● =01 脱机模式。 ● =10 主要模式。 ● =11 次级模式 (备用)。 ● %SW61.14和%SW61.15 来自此PLC的操作模式位 ● =01 脱机模式。 ● =10 主要模式。 ● =11 次级模式 (备用)。	0	无	有
%SW62 WSBFipioDIAG1	显示 Fipio 总线的总线仲裁器功能和生产者 / 消费者功能	最低字节表示生产者 / 消费者功能的状态。 最高字节表示总线仲裁器功能的状态 (BA)。 字节的数值： ● 16#00：该功能不存在 (没有 Fipio 应用程序) ● 16#07：该功能目前处于停止 BA 状态 (已经发送了停止 BA 命令，该命令尚未结束) ● 16#0F：该功能目前处于运行 BA 状态 (已经发送了运行 BA 命令，该命令尚未结束) ● 16#70：该功能已经被初始化，但是还不能操作 (在停止 BA 模式) ● 16#F0：该功能目前被正常执行 (在运行 BA 模式)	0	无	有

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW63 WSDUALDIAG	在 PLC 之间交换诊断字	在 %SW63 到 %SW65 字中可以使用双重 PLC 备份诊断功能。 正常 PLC 字 %SW63， %SW64 和 %SW65 分别包含备份 PLC 字 %SW60， %SW61 和 %SW62。 与此类似，备份 PLC 字 %SW63， %SW64 和 %SW65 分别包含正常 PLC 字 %SW60，%SW61 和 %SW62。 图例 这些字通过 Ethway PLC 间的链接 (TSX ETY 110 模块) 进行交换。	0	无	有
%**SW62** HSBY_REVERSE0 %**SW63** HSBY_REVERSE1	传输字	用户可以在主任务的第一个代码段对这两个字进行写操作。 接下来它们会从保留处理器自动进行传输，以更新主 PLC。它们可以在主 PLC 上读出，用作应用程序的参数。	0	无	有

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW66 WSBSEVENSEG	备份 PLC 诊断	诊断字表示： ● %SW66.0=0 降低的功能 ● %SW66.0=1 名义上的功能 ● %SW66.1=1 PLC A 是正常 PLC ● %SW66.2=1 PLC B 是正常 PLC ● %SW66.3=1 PLC 间的通信错误 **PLC A 上的信息** ● %SW66.4=1 Fipio 网络在 PLC A 上发生严重错误 ● %SW66.5=1 PLC A 处于停止模式 ● %SW66.6=1 PLC A 处于暂停模式 ● %SW66.7=1 在 PLC A 上发生以太网 TCP/IP 通信 错误 (TSX ETY 210 模块或者客户端功能) ● %SW66.8=1 至少有一个 PLC A 机架模块发生错误 ● %SW66.9=1 至少有一个 PLC A 的 Fipio 设备发生错误 **在 PLC B 上的信息** ● %SW66.10=1 在 PLC B 上的 Fipio 网络发生严重错误 ● %SW66.11=1 PLC B 处于停止模式 ● %SW66.12=1 PLC B 处于暂停模式 ● %SW66.13=1 在 PLC B 上发生以太网 TCP/IP 通信 错误 (TSX ETY 210 模块或者客户端功能) ● %SW66.14=1 至少有一个 PLC B 机架模块发生错误 ● %SW66.15=1 至少有一个 PLC B 的 Fipio 设备发生错误	0	无	有
%SW67 WSBADDR	网络地址和双重 PLC 站地址	这个字包含了网络地址和双重 PLC 站地址，可以用来设置 PLC 间的通信。这个字必须以十六进制显示，以便于通过如下方式进行翻译： 最高 / 最低 网络地址 | 工作站地址	-	无	有

字符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW68 WSBTIMEBASE0 **%SW69** WSBTIMEBASE1	计时器功能块所使用的时间基准	这些字包含了一个由基本计时器功能所使用的时间基准。它从正常 PLC 传输到备份 PLC，以便于进行更新和同步化操作。	-	无	有

系统字 %SW70 到 %SW99 描述

详细的描述

系统字 %SW70 到 %SW99 描述：

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW70 WEEKOFYEAR	实时时钟功能	系统字包含一年中的星期的数量：1 到 52。	-	有	有
%SW71 KEY_SWITCH	Quantum 前面板上的开关的位置	这个字给出了 Quantum 处理器前面板上的开关的位置。它会被系统自动更新。 ● %SW71.0 = 1 在 “保护内存” 位置。 ● %SW71.1 = 1 在 “停止” 位置。 ● %SW71.2 = 1 在 “开始” 位置。 ● %SW71.8 = 1 在 “MEM” 位置。 ● %SW71.9 = 1 在 “ASCII” 位置。 ● %SW71.10 = 1 在 “RTU” 位置。 ● %SW71.3 -7 没有使用。	0	有	无
%SW75 TIMEREVTNB	计时器类事件计数器	这个字包含了在队列中的计时器类事件的数量。 (1) 在以下处理器中不可用： TSX 57 1•/2•/3•/4•/5• 。	0	有	有 (1)
%SW76 DLASTREG	诊断功能：记录	上一次登记的结果 ● =0 如果记录成功 ● = 1 如果诊断缓冲器没有配置 ● = 2 如果诊断缓冲器是满的	0	有	有
%SW77 DLASTDEREG	诊断功能：不记录	上一次取消登记的结果 ● =0 如果不记录操作成功 ● = 1 如果诊断缓冲器没有配置 ● = 21 如果错误标识符无效 ● = 22 如果错误没有被记录	0	有	有

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW78 DNBERRBUF	诊断功能：错误的数量	当前在诊断缓冲器中的错误数量	0	有	有
%SW80 MSGCNT0 **%SW81** MSGCNT1 **%SW82** MSGCNT2 **%SW83** MSGCNT3	消息管理	● %SW80：系统发送给终端口的消息数量。 ● %SW81：系统从终端口接收的消息数量。 %SW82：系统发送给 PCMCIA 模块的消息数量。 ● %SW83：系统从 PCMCIA 模块接收的消息数量。 (1) 字 %SW82 和 %SW83 在 Quantum 上不可用。	0	有 (1)	有
%SW84 MSGCNT4 **%SW85** MSGCNT5 **%SW86** MSGCNT6	电报管理	● %SW84：系统发送的电报数量。 ● %SW85：系统接收的电报数量。 ● %SW86：系统拒绝的电报数量。	0	无	有
%SW87 MSTSERVCNT **%SW88** ASNSERVCNT **%SW89** APPSERVCNT	通信流管理	● %SW87：同步服务器在每个主 (MAST) 任务周期内所处理的请求数量。 ● %SW88：异步服务器在每个主 (MAST) 任务周期内所处理的请求数量。 ● %SW89：服务器功能 (即时) 在每个主 (MAST) 任务周期内所处理的请求数量。 (1) 字 %SW88 和 %SW89 在 Quantum 上不可用。	0	有 (1)	有

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW90 MAXREQNB	在每个主任务周期内所处理的最大请求数量	这个字用来设定 PLC 在每个主任务周期内所处理的最大请求 (包括所有协议：UNI-TE，Modbus，等等) 数量。 这个最大请求数量必须在 2 到 (N+4) 之间。 N：具体的数量取决于模块。 ● TSX 57 1• ：N = 4 ● TSX 57 2• ：N = 8 ● TSX 57 3• ：N = 12 ● TSX 57 4• ：N = 16 ● 140 CPU 31••/43••/53••/： N = 5 TSX 57 5• 和 140 CPU 6••：N = 20 对于 Quantum 140 CPU 31 •• /43 •• /53 •• / ：这个值会被设定，相当于 5。如果这个字为缺省值 0，或者输入值超出了范围，会采纳数值 N。	0	有	有
%SW99 ADR/SWAP	通信冗余管理	这个字用来管理网络模块的冗余。 如果用来访问 x 号网络 (X-WAY) 的通信模块被检测出了问题，可以通过在 %SW99 字中输入网络号来转换到另外一个通信模块 (连接到同一个网络上)。 系统可以将 %SW99 字复位为 0。	a	无	有

系统字 %SW108 到 %SW116 描述

详细的描述

系统字 %SW108 到 %SW116 描述：

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW108 FORCEDIOIM	强制的 I/O 模块位数量	该系统字计算强制的 I/O 模块位数量。该位随着每一次强制操作而递增，随着每一次解除强制操作而递减。	0	有	有
%SW109 FORCEDANA	强制的模拟通道数量	该系统位计算强制模拟通道的数量。该位随着每一次强制操作而递增，随着每一次解除强制操作而递减。	0	无	有
%SW116 REMIOERR	任务中的 Fipio I/O 错误	正常情况下被设置为 0，这个字的每一位都显示了正在被测试的**任务内**的 Fipio 交换状态。用户可以把这个字设置为 0。关于 %SW116 字所含各位的详细情况： ● %SW116.0 = 1 显式交换错误 (变量没有在总线上进行交换) ● %SW116.1 = 1 在显式交换中超时 (在定义的时间结束时候没有应答) ● %SW116.2 = 1 在同一时间内达到了最大显式交换数量 ● %SW116.3 = 1 某个帧无效 ● %SW116.4 = 1 接收的帧的长度大于声明的帧的长度 ● %SW116.5 = 保留位，置为 0 ● %SW116.6 = 1 某个帧无效，或者某个代理正在初始化 ● %SW116.7 = 1 缺失了一个配置设备 ● %SW116.8 = 1 通道错误 (至少一个设备通道给出错误) ● %SW116.9 to 15 = 保留位，置为 0	-	无	有

系统字 %SW124 到 %SW127 描述：

详细的描述

系统字 %SW124 到 %SW127 描述：

字 符号	功能	描述	初始 状态	Quantum	Premium Atrium
%SW124 CPUERR	系统错误类型	系统将上一次遇到的系统错误类型写入这个字中 (在冷重启的过程中，这些代码不变)： ● 16#30 ：系统代码错误。 ● 16#53 ：在 I/O 交换过程中发生超时错误。 ● 16#60 to 64：堆栈溢出。 ● 16#90 ：系统转换错误：不可预知的 IT。	-	有	有
%SW125 BLKERRTYPE	模块化错误类型	系统将上一次遇到的模块化错误类型写入这个字中： ● 16#2XXX：针对一个未定义的子程序执行一个调用指令。 ● 16#0XXX：执行一个未知功能。 ● 16#2258 ：执行暂停指令。 ● 16#DE87：浮点数运算错误。 ● 16#DEB0：监视时钟溢出。 ● 16#DEF0：除数为 0。 ● 16#DEF1：字符串传输错误。 ● 16#DEF2：容量溢出。 ● 16#DEF3：索引溢出。 ● 16#DEF7：SFC 执行错误。 ● 16#DEF8：针对一个未定义的选项卡执行 JMP 指令。 ● 16#DEFE：SFC 程序含有针对未定义步的交叉引用。 ● 16#DEFF：没有实现浮点功能。 **注意：**浮点运算错误会在 %SW17 字中表示出来。	-	有	有

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW126 ERRADDR0 **%SW127** ERRADDR1	指令地址的模块化错误	生成应用程序模块的指令地址发生错误。 对于 16 位处理器，TSX P57 1•• /2 •• • %SW126 包含该地址的偏移量。 • %SW127 包含该地址的区段号。 对于 32 位处理器： • %SW126 包含用于该地址的最低字。 • %SW127 包含用于该地址的最高字。	0	有	有

6.3 Atrium/Premium - 专用系统字

描述

本节主题

本节描述了 Premium 和 Atrium PLC 的系统字 %SW128 到 %SW167。

本节内容

本节包含以下主题：

系统字 %SW128 到 %SW143 描述

详细的描述

系统字 %SW128 到 SW143 描述：

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW128...143 ERRORCNXi 其中 i = 0 到 15	错误的 Fipio 连接点	在这组字中的每一位都代表与 Fipio 总线相连的一个设备的状态。它们通常情况下被设为 1，如果其中的某一位为 0，就表示该连接点发生了错误。对于一个未经配置的连接点来说，相应的位一直是 1。	0	无	有

下面的表格给出了字的各位与连接点地址之间的关系：

	第 0 位	第 1 位	第 2 位	第 3 位	第 4 位	第 5 位	第 6 位	第 7 位	第 8 位	第 9 位	第 10 位	第 11 位	第 12 位	第 13 位	第 14 位	第 15 位
%SW128	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
%SW129	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
%SW130	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
%SW131	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
%SW132	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
%SW133	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
%SW134	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
%SW135	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
%SW136	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
%SW137	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
%SW138	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
%SW139	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
%SW140	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
%SW141	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
%SW142	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
%SW143	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

系统字 %SW144 到 %SW146 描述

详细的描述

系统字 %SW144 到 %SW146 描述：

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW144 BAOPMOD	Fipio 总线仲裁器功能操作模式	这个系统字用来启动和停止总线仲裁器功能和生产者 / 消费者功能。它可以在停止状态下可修改为启动，自动和手工模式。 ● %SW144.0 ● = 1：在运行模式下的生产者 / 消费者功能。 ● = 0：在停止模式下的生产者 / 消费者功能 (在总线上没有变量交换)。 ● %SW144.1 ● = 1：总线仲裁器处于运行模式。 ● = 0：总线仲裁器处于停止模式 (在总线上没有变量或者信息扫描)。 ● %SW144.2 ● = 1：在自动总线停止状态下进行自动启动。 ● = 0：在自动总线停止状态下进行手工启动。 %SW144.3 到 15 为保留位，置为 0。	0	无	有

字 符号	功能	描述	初始 状态	Quantum	Premium Atrium
%SW145 BAPARAM	修改 Fipio 总线仲裁器参数	该位由用户置为 1，在初始化的过程中被系统复位为 0。 ● %SW145.0 = 1：修改总线仲裁器的优先级；这个系统字的最高字节包含了要用于总线的总线仲裁器优先级数值。 ● %SW145.1 = 1：修改时隙 (Tr) 的数值；这个系统字的最高字节包含了要用于总线的 Tr (用毫秒来表示) 数值。在初始化结束以后，这个系统字会被复位为 0。 ● %SW145.2 = 1：修改暂停时间 (T0) 的数值；这个系统字的最高字节包含了要用于总线的 T0 (用毫秒来表示) 数值。在初始化结束以后，这个系统字会被复位为 0。 ● %SW145.3 到 %SW145.7 为保留位，被置为 0。 ● %SW145.8 到 %SW145.15 = 1: 根据第 0，1，和 2 位的数值，该字节包含了要用于总线的数值。 可以在总线仲裁器处于运行模式的时候修改这些参数，但是如果要在应用程序中使用它们，必须停止总线仲裁器，然后重新启动。	0	无	有
%SW146 BASTATUS	Fipio 总线仲裁器功能显示	最低字节表明了生产者 / 消费者功能的状态。 最高字节表明了总线仲裁器功能的状态。 字节数值： ● 16#00：该功能不存在 (没用 Fipio 应用程序)。 ● 16#70 ：该功能已经被初始化，但是没有用 (处于停止模式)。 ● 16#F0：该功能当前被正常执行 (处于运行模式)。	0	无	有



警告

关于 %SW144 和 %SW145 字

如果修改这些系统字，会使 PLC 工作站停止。

如果未能遵守此处的警告，可能会造成伤害或者设备损坏。

系统字 %SW147 到 %SW152 描述

详细的描述

系统字 %SW147 到 %SW152 描述：

字 符号	功能	描述	初始状态	Quantum	Premium
%SW147 TCRLIBRE	MAST 网络扫描时间	如果数值不是 0，就代表 MAST 任务网络扫描时间 (TCR-MAST) 的数值 (用毫秒表示)。	0	无	有
%SW148 TCR1ASSERV	MAST 网络扫描时间	如果数值不是 0，就代表第一个 MAST 任务网络扫描时间 (TCR-MAST) 的数值 (用毫秒表示)。	0	无	有
%SW149 TCR2ASSERV	MAST 网络扫描时间	如果数值不是 0，就代表第二个 MAST 任务网络扫描时间 (TCR-MAST) 的数值 (用毫秒表示)。	0	无	有
%SW150 NBFRSENT	发送的帧的数量	这个字代表由 Fipio 通道管理器所发送的帧的数量。	0	无	有
%SW151 NBFRREC	接收的帧的数量	这个字代表由 Fipio 通道管理器所接收的帧的数量。	0	无	有
%SW152 NBRESENTMSG	重新发送的消息的数量	这个字代表由 Fipio 通道管理器重新发送的消息的数量。	0	无	有

系统字 %SW153 描述

详细的描述

系统字 %SW153 描述：

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW153 FipioERR0	Fipio 通道管理器错误列表	每一位都被系统置为 1，被用户置为 0。 详细内容请参见下面的列表。	0	无	有

位的描述

- 第 0 位 = “站上溢错误”：表示在接收的过程中丢失 MAC 符号- 与接收器相链接的站反应太慢。
- 第 1 位 = “消息拒绝错误”：表示接收了被拒绝的确认消息，或者接收的 MAC 根本没有经过确认。
- 第 3 位 = “站下溢错误”：表示站无法按希望速度在网络上实现传输。
- 第 4 位 = “物理层错误”：表示在物理层缺少延时发射功能。
- 第 5 位 = “无反应错误”：表示发射器已经正常发射信号，并且符合操作范围的发射电流，但是，在同一个通道内没有检测到返回信号。
- 第 6 位 = “对话错误”：表示发射器当前控制线路的时间超过了系统设定的最大操作范围，从而发生错误。这个错误可能是由于调节器性能退化，或者错误的数据链接层所引起的。
- 第 7 位 = “电流不足的错误”：表示发射器在收到请求以后所产生的电流小于系统设定的最小操作范围，从而发生错误。这个错误是由增加的阻抗引起的 (比如开路等情况)。
- 第 8 位 = “帧被分开的错误”：表示帧的起始定界符已经被识别，但是结尾定界符尚未被识别，此时在帧体内接收到暂停信息。在正常操作模式下，在结尾定界符被识别以后，暂停信息才会出现。
- 第 9 位 =“接收帧 CRC 错误”：表示在正常接收的帧上所计算的 CRC 和在该帧内所包含的 CRC 具有不同的数值。
- 第 10 位 = “接收帧代码错误”：表示专属于帧头和帧尾定界顺序的符号出现在帧体中。
- 第 11 位 = “接收帧长度错误”：收到的帧体长度超过了 256 个字节。
- 第 12 位 = “接收到了未知的帧类型”：在帧体内，第一个字节标明帧链接的类型。在 WorldFip 的标准链接协议中定义了一系列帧类型。在帧中所发现的任何其他代码都属于未知帧类型。
- 第 13 位 = “接收到了一个被删剪的帧”：目标站在等待帧头的过程中，识别出标定帧尾的顺序符号。
- 第 14 位 = “未使用，无意义的数值”。
- 第 15 位 = 未使用，无意义的数值

系统字 %SW154 描述

详细的描述

系统字 %SW154 描述：

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW154 FipioERR1	Fipio 通道管理器错误列表	每一位都被系统设置为 1，被用户设置为 0。 详细内容请参见下面的列表。	0	无	有

位的描述

- 第 0 位 = “非周期性顺序超时”：表示消息或非周期性变量窗口在宏周期超过了一个基本周期的范围。
- 第 1 位 = “拒绝消息请求”：表示消息队列已经饱和 - 目前总线仲裁器无法锁定或者响应一个请求。
- 第 2 位 = “紧急更新命令被拒绝”：表示用于紧急非周期性变量交换请求的队列已经饱和 - 目前总线仲裁器无法锁定或者响应一个请求。
- 第 3 位 = “非紧急更新命令被拒绝”：表示用于非紧急非周期性变量交换请求的队列已经饱和 - 目前总线仲裁器无法锁定或者满足一个请求。
- 第 4 位 = “暂停错误”：总线仲裁器在大于标准 WorldFip 时间周期的时间周期内没有检测到任何总线信息。
- 第 5 位 = “在发送标识符的时候发生网络冲突”：表示在理论暂停周期内发生了网络活动。在发送和等待总线仲裁器回复期间，总线上应该没有任何信息。如果总线仲裁器检测到了信息，它会生成一个冲突错误 (比如当总线上有若干个仲裁器同时有效时)。
- 第 6 位 = “总线仲裁器溢出错误”：表示在访问总线仲裁器站内存的时候出现了冲突。
- 第 7 位 = “未使用，无意义的数值”。
- 第 8 到第 15 位 = 保留位，置为 0。

Premium/Atrium 系统字 %SW155 到 %SW167 描述

详细的描述

系统字 %SW155 到 %SW167 描述：

字 符号	功能	描述	初始 状态	Quantum	Premium Atrium
%SW155 NBEXPLFIP	在 Fipio 上的显式交换的数量	当前正在 Fipio 上处理的，通过指令 (READ_STS，REA_PARAM，等等) 执行的显式交换的数量。 也包括通过请求 (READ_IO_OBJECT，WRITE_IO_OBJECT，等等) 执行的显式交换的数量。 **注意：**显式交换的数量总是小于 24。	0	无	有
%SW160 to %SW167 PREMRACK0 to PREMRACK7	PLC 模块的操作状态	字 %SW160 到 %SW167 分别与机架 0 到 7 相关联。 这些字的 0 到 15 位则与位于这些机架的 0 到 15 位置的模块相关联。 如果模块有错误，相应的位会被置为 0 ；如果模块正常，相应的位会被置为 1。 **例子：**%SW163.5 =0 位于第 3 个机架第 5 个插槽的模块有错误。	0	无	有

6.4 Quantum- 专用系统字

描述

本节主题

本节描述了 Quantum PLC 的系统字 %SW128 到 %SW547。

本节内容

本节包含以下主题：

Quantum 系统字 %SW128 到 %SW179 描述

详细的描述

系统字 %SW128 到 %SW179 描述；这些字在 Quantum 140 CPU 6 ••• PLC 上是有效的：

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW128 NB_P502_CNX	打开的连接的数量	这个字的最高字节表示在以太网链接 TCP/IP 端口 502 上打开的 TCP 连接的数量。	0	有	无
%SW129 NB_DENIED_CNX	被拒绝连接的数量	这个字表示在以太网链接 TCP/IP 端口 502 上被拒绝的 TCP 连接的数量。	0	有	无
%SW130 NB_P502_REF	被拒绝的消息的数量	这个字表示在以太网链接 TCP/IP 端口 502 上被拒绝的 TCP 消息的数量。	0	有	无
%SW132 and **%SW133** NB_SENT_MSG	发送的消息的数量	这个双字 %SDW132 表示在以太网链接 TCP/IP 端口 502 上发送的消息的数量。	0	有	无
%SW134 and **%SW135** NB_RCV_MSG	接收的消息的数量	这个双字 %SDW134 表示在以太网链接 TCP/IP 端口 502 上接收的消息的数量。	0	有	无
%SW136 NB_IOS_CNX	扫描的设备的数量	这个字表示在以太网链接 TCP/IP 端口 502 上扫描的设备的数量。 .	0	有	无
%SW137 NB_IOS_MSG	接收到 IO 扫描消息的数量	这个字表示在以太网链接 TCP/IP 端口 502 上每秒钟从 IO 扫描设备所接收的消息的数量。	0	有	无
%SW138 GLBD_ERROR	全球数据一致性错误	全球数据一致性错误	0	有	无

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW139 BW_GLBD_IOS	全局数据和 IO 扫描设备负载	这个字的低字节表示与 IO 扫描相关的负载百分比。 这个字的高字节表示与全局数据相关的负载百分比。	0	有	无
%SW140 BW_OTHER_MSG	发送消息的设备和其他设备的负载	这个字的低字节表示与发送消息相关的负载百分比。 这个字的高字节表示与其他设备相关的负载百分比。	0	有	无
%SW141 and %SW142 IP_ADDR	IP 地址	这个双字 %SDW141 接收以太网链接的 IP 地址。	0	有	无
%SW143 and %SW144 IP_NETMASK	IP 子网掩码	这个双字 %SDW143 接收以太网链接的子网掩码。	0	有	无
%SW145 and %SW146 IP_GATEWAY	缺省的以太网网关地址	这个双字 %SDW145 接收缺省的以太网网关地址。	0	有	无
%SW147 to %SW149 MAC_ADDR1 to 3	MAC 地址	字 %SW147， %SW148 和 %SW149 分别代表 MAC 1， MAC 2 和 MAC 3 地址。	0	有	无
%SW150 FW_VERSION	嵌入软件版本	这个字代表嵌入软件版本。	0	有	无

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW151 BOARD_STS	以太网链接的状态	这个字代表以太网链接的状态。 ● 第 0 位 =0，如果以太网链接被停止 ● 第 1 位 =0 ● 第 2 位：0= 半双工模式， 1= 全双工模式 ● 第 3 位 =0 ● 第 4 位到第 11 位：=7 用于 Quantum，=6 用于 Hot Standby Quantum ● 第 12 位：0 = 10 兆比特链接，1= 100 兆比特链接 ● 第 13 位：0 = 10/100Base-TX 链接 (双绞线) ● 第 14 位：0 ● 第 15 位：0 = 以太网链接失效，1= 以太网链接有效	0	有	无
%SW160 to %SW167 REFRESH_IO	由 IO 扫描决定的设备操作状态	字 %SW160 到 %SW167 的各位与被 IO 扫描的设备相关联。 如果设备出错，相应的位会被设为 0 ；如果设备正常操作，相应的位就会被设为 1。 %SW160.0：1 号设备。 %SW160.1：2 号设备。 .... %SW167.15：128 号设备。	-	有	无
%SW168 to %SW171 VALID_GD	全局数据的操作状态	字 %SW168 to %SW171 的各位与全局数据相关联。 如果设备出错，相应的位会被设为 0 ；如果设备正常操作，相应的位就会被设为 1。 %SW168.0：1 号设备。 %SW168.1：2 号设备。 .... %SW171.15：64 号设备。	-	有	无

Quantum 系统字 %SW180 到 %SW640 描述

详细的描述

系统字 %SW180 到 %SW640 描述：

字 符号	功能	描述	初始状态	Quantum	Premium Atrium

%SW180 to %SW339
`IOHEALTHij`
`i=1..32,`
`j=1..5`
	PLC 模块的操作状态	字 %SW180 到 %SW339 与 PLC 站相关联：每个站有 5 个字，对应着站的第 1 到第 5 个机架。
		
		%SW180：第一个站第一个机架的模块操作状态。
		%SW181：第一个站第二个机架的模块操作状态。
		....
		%SW185：第二个站第一个机架的模块操作状态。
		%SW186：第二个站第二个机架的模块操作状态。
		....
		
		这些字的 0 到 15 位对应着机架上第 16 个位置到第 1 个位置的各个模块。
		
		如果模块出错，相应的位会置为 0 ；如果模块正常操作，相应的位就会置为 1。
		
		例子： %SW185.5 =0
		位于第 2 个远程站第 1 个机架第 11 个插槽内的模块出错。
			0	有	无

%SW340 `MB+DIOSLOT`	带有 Modbus Plus 链接的处理器的插槽号	带有用于连接第一个 DIO 网络的内置 Modbus Plus 链接处理器的插槽号。插槽号从 0 到 15。	-	有	无

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW341 to %SW404 MB+IOHEALTHi i=1..64	第一个 DIO 网络的分布式站模块的操作状态	字 %SW341 到 %SW404 与分布式站 (DIO) 相关联：64 个字对应着第一个网络中的 64 个 DIO 站。 %SW341：第 1 个站的模块的操作状态。 %SW342：第 2 个站的模块的操作状态。 .... %SW404：第 64 个站的模块的操作状态。 这些字的 0 到 15 位对应着站上第 16 个位置到第 1 个位置的各个模块。 如果模块出错，相应的位会置为 0 ；如果模块正常操作，相应的位就会置为 1。 **例子：** %SW362.5 =0 位于第 1 个 DIO 网络内的第 22 个工作站第 11 个插槽中的模块出错。	-	有	无
%SW405 Œ?M1DIOSLOT	DIO 网络第一个接口模块的插槽号	用于连接第二个 DIO 网络的 140 NOM 2 •• 模块的插槽号。插槽号从 0 到 15。	-	有	无

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW406 to %SW469 Œ?M1DIOHEALTHi i=1..64	第二个 DIO 网络的分布式站模块的操作状态	字 %SW406 到 %SW469 与分布式站 (DIO) 相关联：64 个字对应着第二个网络中的 64 个 DIO 站。 %SW406：第 1 个站的模块的操作状态。 %SW407：第 2 个站的模块的操作状态。 .... %SW469：第 64 个站的模块的操作状态。 这些字的 0 到 15 位对应着工作站上第 16 个位置到第 1 个位置的各个模块。 如果模块出错，相应的位会置为 0 ；如果模块正常操作，相应的位就会置为 1。 **例子：** %SW412.5 =0 位于第 2 个 DIO 网络内的第 7 个站第 11 个插槽中的模块出错。	-	有	无
%SW470 Œ?M2DIOSLOT	DIO 网络第二个接口模块的插槽号	用于连接第三个 DIO 网络的 140 NOM 2 •• 模块的插槽号。插槽号从 0 到 15。	-	有	无

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW471 to %SW534 Œ?M2DIOHEALTHi i=1..64	第三个 DIO 网络的分布式站模块的操作状态	字 %SW471 到 %SW534 与分布式站 (DIO) 相关联：64 个字对应着第三个网络中的 64 个 DIO 站。 %SW471：第 1 个站的模块的操作状态。 %SW472：第 2 个站的模块的操作状态。 .... %SW534：第 64 个站的模块的操作状态。 这些字的 0 到 15 位对应着站上第 16 个位置到第 1 个位置的各个模块。 如果模块出错，相应的位会置为 0 ；如果模块正常操作，相应的位就会置为 1。 **例子：** %SW520.5 =0 位于第 3 个 DIO 网络内的第 50 个站第 11 个插槽中的模块出错。	-	有	无

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW535 RIOERRSTAT	RIO 启动错误	这个字存储了启动错误代码。在系统运行的时候，这个字会一直设置 0 ；如果发生了错误，PLC 不会启动，而是产生一个停止状态代码 01：I/O 分配长度 02：远程 I/O 链接号码 03：在 I/O 分配中的站号 04：I/O 分配校验和 10：站描述符长度 11：I/O 站号码 12：工作站自治时间 13：ASCII 端口号 14：站模块的数量 15：已经配置的站 16：已经配置的端口 17：超过 1024 个输出点 18：超过 1024 个输入点 20：模块插槽地址 21：模块机架地址 22：输出字节的数量 23：输入字节的数量 25：第一个参考号 26：第二个参考号 28：超出 16 位范围的内部位 30：不成双的单个输出模块 31：不成双的单个输入模块 32：不成双的单个模块参考 33：在寄存器 3x 后面的参考 1x 34：已经使用的虚设模块参考 35：模块 3x 不是一个虚设模块 36：模块 4x 不是一个虚设模块	-	有	无

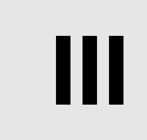
字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW536 CAERRCNT0 **%SW537** CAERRCNT1 **%SW538** CAERRCNT2	线缆 A 的通信状态	字 %SW536 到 %SW538 是线缆 A 的通信错误字。 ● %SW536： 　● 高字节：帧错误计数。 　● 低字节：DMA 接收器的计数溢出。 ● %SW537： 　● 高字节：接收器错误计数。 　● 低字节：错误站接收计数。 ● %SW538： 　● %SW538.15 = 1，短帧 　● %SW538.14 = 1，没有尾帧字符 　● %SW538.3 = 1，CRC 错误 　● %SW538.2 = 1，对齐错误 　● %SW538.1 = 1，溢出错误 　● %SW538.13 到 4 和 0 未使用	-	有	无
%SW539 **CBERRCNT0** **%SW540** **CBERRCNT1 to** **%SW541** CBERRCNT2	线缆 B 的通信状态	字 %SW539 到 %SW541 是线缆 B 的通信错误字。 ● %SW539： 　● 高字节：帧错误计数。 　● 低字节：DMA 接收器的计数溢出。 ● %SW540： 　● 高字节：接收器错误计数。 　● 低字节：错误站接收计数。 ● %SW541： 　● %SW541.15 = 1，短帧 　● %SW541.14 = 1，没有帧尾 　● %SW541.3 = 1，CRC 错误 　● %SW541.2 = 1，对齐错误 　● %SW541.1 = 1，溢出错误 　● %SW541.13 到 4 和 0 未使用	-	有	无

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW542 **GLOBERRCNT0** **%SW543** **GLOBERRCNT1** **%SW544** GLOBERRCNT2	全局通信状态	字 %SW542 到 %SW544 是全局通信错误字。 ● %SW542：显示了全局通信状态。 ● %SW542.15 = 1，通信操作正确 ● %SW542.14 = 1，在线缆 A 上的通信操作正确 ● %SW542.13 = 1，在线缆 B 上的通信操作正确 ● %SW542.11 到 8= 通信计数器丢失 ● %SW542.7 到 0 = 重试总数计数 ● %SW543：用于线缆 A 的全局错误总数计数： ● 高字节：检测到的错误计数 ● 低字节：“非响应”计数 ● %SW544：用于线缆 B 的全局错误总数计数： ● 高字节：检测到的错误计数 ● 低字节：“非响应”计数	-	有	无

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW545 to %SW640 MODUNHEALTHi IOERRCNTi IORETRYi (i=1..32)	远程站的状态	字 %SW545 到 %SW640 用来描述远程站的状态。每一个远程站使用三个状态字。 ● %SW545： 显示第一个远程站的全局通信状态。 　● %SW545.15 = 1，通信操作正确 　● %SW545.14 = 1，在线缆 A 上的通信操作正确 　● %SW545.13 = 1，在线缆 B 上的通信操作正确 　● %SW545.11 到 8= 通信计数丢失 communications counter 　● %SW545.7 到 0 = 重试总数计数 ● %SW546：是用于第一个远程站线缆 A 的全局错误总数计数： 　● 高字节：检测到的错误计数 　● 低字节： “非响应”计数 ● %SW547：是用于第一个远程站线缆 B 的全局错误总数计数： 　● 高字节：检测到的错误计数 　● 低字节： “非响应”计数 字： %SW548 到 550 分配给第 2 个远程站 %SW551 到 553 分配给第 3 个远程站 .... %SW638 到 640 分配给第 32 个远程站	-	有	无

字 符号	功能	描述	初始状态	Quantum	Premium Atrium
%SW545 to %SW547 MODUNHEALTH1 IOERRCNT1 IORETRY1	本地工作站的状态	对于第 1 个工作站预留给本地输入 / 输出的 PLC 来说，状态字 %SW545 到 %SW547 用在以下方面。 ● %SW545：本地工作站的状态。 ● %SW545.15 = 1，所有模块都正确操作。 ● %SW545.14 到 8= 未使用，总是设置为 0。 ● %SW545.7 到 0 = 模块出现缺陷的次数；计数器在 255 的之候返回。 ● %SW546： 它用作 16 位输入 / 输出总线错误计数器。 ● %SW547： 它用作16位输入/输出总线重复计数器。	-	无	有

数据描述



内容预览

本章内容

本部分描述了可以用在项目中的各种数据类型，以及它们的应用。

本部分内容

本部分包含以下各章：

数据综述

7

内容预览

本章主题

本章给出了以下内容的综述：
- 各种数据类型
- 数据实例
- 数据引用

本章内容

本章包含以下内容：

常规信息

介绍

数据项指定了一个可以被实例化的对象，比如：

- 一个变量，
- 一个功能块。

数据分三个阶段被定义，分别是：

- **数据类型**阶段，说明以下内容：
  - 它的类别，
  - 它的格式。
- **数据实例**阶段，定义它的存储位置和属性，它可以是：
  - 定位的，或者
  - 非定位的。
- **数据引用**阶段，定义它的访问方式：
  - 通过立即值，
  - 通过名称，
  - 通过地址。

图例

下面是定义数据的三个阶段：



在对一个数据项进行**实例化**的时候，要根据它的类型为它分配一个内存插槽。

在对一个数据项进行**引用**的时候，要为它定义一个引用 (名称，地址，等等)，从而允许在内存中访问数据项。

数据类型系列综述

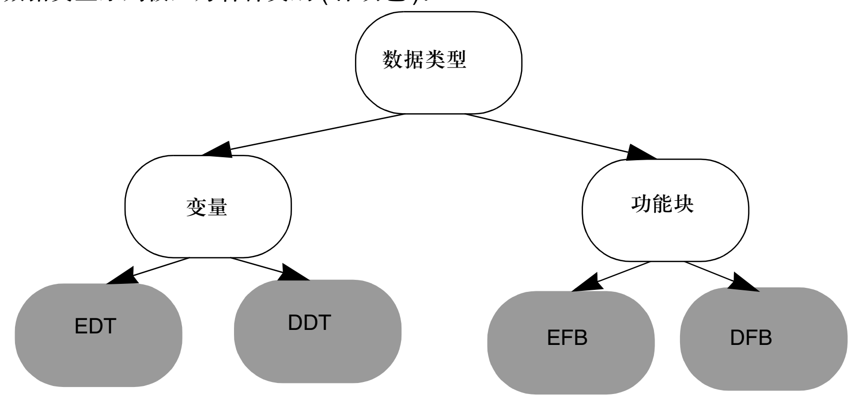
介绍

数据类型是一个软件信息，它为数据项说明以下内容：
- 它的结构
- 它的格式
- 它的属性列表
- 它的行为

这些属性会被该数据类型的所有实例所共享。

图例

数据类型系列被归为各种类别 (深灰色)：



定义

数据类型系列及其定义

系列	定义
EDT	基本数据类型，比如： ● 布尔 ● 整数 ● 字节 ● 字 ● 双字 ● 其他
DDT	导出数据类型，比如： ● 数据表，其中包含同一类型的元素： ● 布尔数据表 (EDT 数据表) ● 数据表的数据表 (DDT 数据表) ● 结构的数据表 (DDT 数据表) ● 结构，其中包括不同类型的元素： ● 布尔结构，字结构，等等 (EDT 结构) ● 数据表的结构，结构的结构，数据表 / 结构的结构 (DDT 结构) ● 布尔结构，数据表结构，等等 (EDT 和 DDT 结构) ● 关于输入 / 输出数据的结构 (IODDT 结构) ● 包含恢复顺序功能图动作或转移状态属性的变量的状态结构
EFB	用 C 语言编写的基本功能块，其中包括： ● 输入变量 ● 内部变量 ● 输出变量 ● 一个处理算法
DFB	用自动化语言 (结构化文本，指令表，等等) 编写的导出功能块，其中包括： ● 输入变量 ● 内部变量 ● 输出变量 ● 一个处理算法

数据实例概述

介绍

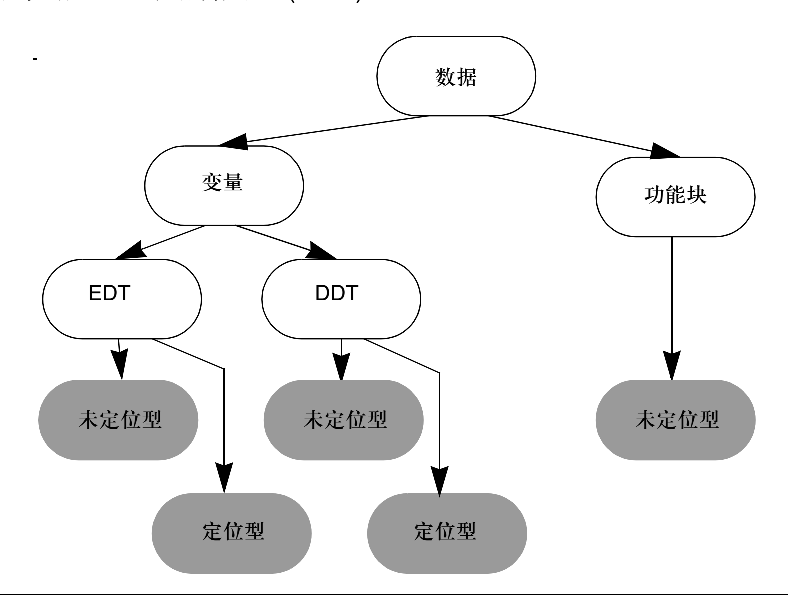
数据实例是一个单独的功能实体，它具有它所属的数据类型的所有特性。

一个数据类型可以拥有一个或多个实例。

数据实例可以有一个如下的内存分配：
- 未定位型
- 或者定位型

图例

隶属于不同类型的实例内存分配 (深灰)：



定义

数据实例内存分配定义：

数据实例	定义
未定位型	系统会自动分配实例的内存槽，它可以随着应用程序的更新而变化。实例通过用户所选择的名称 (符号) 进行定位。
定位型	实例的内存槽是固定的，预先定义的，永远都不能改变。 实例通过用户所选择的名称 (符号) 以及厂商所定义的拓扑地址进行定位，或者仅仅通过厂商所定义的拓扑地址进行定位。

数据引用概述

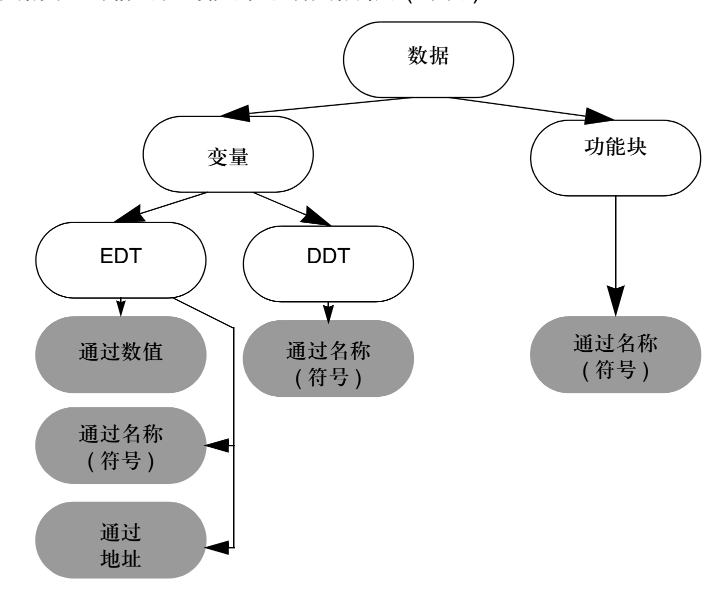
介绍

数据引用允许用户通过以下方式之一访问该数据的实例：

- 立即值，仅对于 EDT 类型的数据是真的
- 地址设置，仅对于 EDT 类型的数据是真的
- 名称 (符号)，对于所有 EDT，DDT，EFB，DFB 数据类型以及 SFC 对象都是真的

图例

根据数据类型的情况，可能出现的数据引用 (深灰)：



类型 \ 实例名称的语法规则

介绍

类型和变量的语法名称可以在有扩展字符集的情况下编写，也可以在没有扩展字符集的情况下编写。该选项可以在工具 **-> 项目设置**菜单中的**语言扩展**选项卡中进行选择。

- 如果选择了**允许扩展字符集**选项，应用程序就遵循 IEC 标准
- 如果没有选择**允许扩展字符集**选项，用户拥有一定的自由度，但是应用程序不遵循 IEC 标准

用于输入到应用程序的名称的扩展字符集涉及到：

- DFB (导出功能块) 用户功能块或者 DDT (导出数据类型)
- 组成 DFB/EFB 功能块数据类型或者导出数据类型 (DDT) 的内部元素
- 数据实例

如果没有选择了 "允许扩展…" 复选框

输入的名称是由字母数字字符和下划线字符所组成的字符串。

相关的规则如下：

- 名称的第一个字符是一个字母数字字符或者一个下划线
- 不能连续使用两个下划线字符

如果选择 "允许扩展…" 复选框

输入的名称是由字母数字字符和下划线字符所组成的字符串。

另外可以使用一些附加字符，比如：

- 对应 ASCII 码 192 到 223 的字符 (除了 215 号)
- 对应 ASCII 码 224 到 255 的字符 (除了 247 号)

相关的规则如下：

- 名称的第一个字符是一个**字母数字**字符或者一个下划线
- **可以**连续使用下划线字符

数据类型

8

内容预览

本章主题

本章描述了可以用于应用程序的所有数据类型。

本章内容

本章包含以下各节：

8.1 二进制格式的基本数据类型 (EDT)

内容预览

本节主题

本节描述了二进制格式的数据类型，它们是：
- 布尔类型
- 整数类型
- 时间类型

本节内容

本节包含以下主题：

二进制格式的数据类型概述

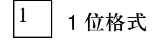
介绍

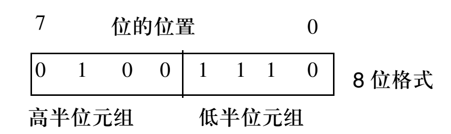
二进制格式的数据类型属于 EDT (基本数据类型) 系列，该系列包含单一数据类型，而不包括导出数据类型 (数据表，结构，功能块)。

关于二进制格式的提示

一个二进制格式的数据项由一位或者多位组成，其中的每一位都用 2 个基数 (0 或 1) 来表示。

数据项的大小由它的位数来决定。比如：





数据项可以是：
- 有符号的。最高位是符号位：
  - 0 表示正数
  - 1 表示负数

数值的范围是：

$$[-2^{\langle 位-1\rangle}, 2^{\langle 位-1\rangle}-1]$$

- 无符号的。所有的位都用来表示数值

数值的范围是：

$$[0, 2^{位}-1]$$

位 = 位数 (格式)。

二进制格式的数据类型

数据类型列表：

类型	名称	格式 (位)	缺省值
BOOL	布尔	8	0=(False)
EBOOL	布尔，带有强制和边缘检测	8	0=(False)
INT	整数	16	0
DINT	双整数	32	0
UINT	无符号整数	16	0
UDINT	无符号双整数	32	0
TIME	无符号双整数	32	T=0s

布尔类型

内容预览

有两种布尔类型：

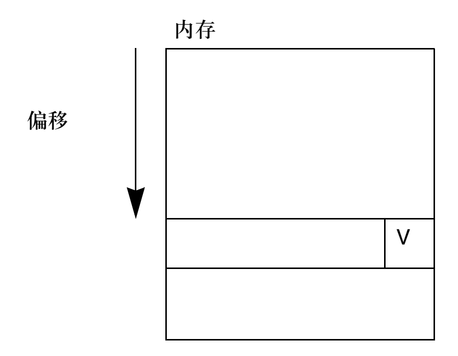
- BOOL 类型，只包含数值 FALSE (=0) 或者 TRUE (=1)
- EBOOL 类型，包含数值 FALSE (=0) 或者 TRUE (=1)，另外还包含关于对上升沿或下降沿以及强制进行管理的信息

BOOL 类型的原则

该类数据占据一个内存字节，但是其数值只存储在一位中。

可以通过一个含有相关字节偏移的地址对该类数据进行访问：

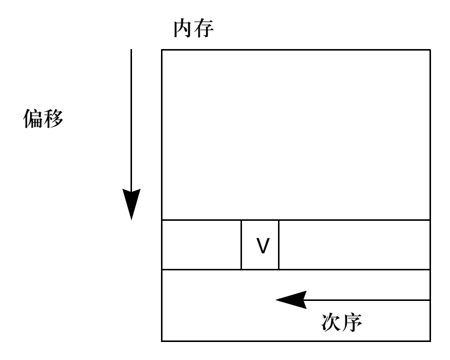
地址设置：



如果相关的**位是从字中提取**的，可以通过一个包含如下信息的地址对它进行访问：

- 相关字节的偏移
- 定义它在字中的位置的次序信息

地址设定：

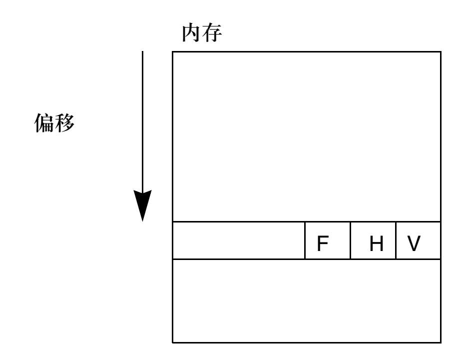


EBOOL 类的原则

该类数据占据一个内存字节。

在对该类数据进行操作的过程中，有三位是必需的：
- 包含当前状态的位 (V)
- 包含以前状态的位 (H)
- 包含强制状态的位 (F)

可以通过一个给定相关字节偏移的地址来访问该类数据：
地址设定：



属于布尔类型的 PLC 变量

变量列表

变量	类型
内部字	EBOOL
系统字	BOOL
字提取位	BOOL
%I 输入	
模块错误位	BOOL
通道错误位	BOOL
输入位	EBOOL
%Q 输出	
输出位	EBOOL

BOOL 和 EBOOL 的兼容性

可以在这两种变量之间进行的操作包括：
- 数值复制
- 地址复制

在类型之间进行复制

	BOOL 目标	**EBOOL 目标**
BOOL 源	是	是
EBOOL 源	是	是

在基本功能 (EF) 参数之间的兼容性

有效参数 **(EF 外部)**	**形式 BOOL 参数** **(EF 内部)**	**形式 EBOOL 参数** **(EF 内部)**
BOOL	是	否
EBOOL	In -> 是 In-Out -> 否 Out -> 否	是

在功能块 (EFB\DFB) 参数之间的兼容性

有效参数 **(FB 外部)**	**形式 BOOL 参数** **(FB 内部)**	**形式 EBOOL 参数** **(FB 内部)**
BOOL	是	In -> 是 In-Out -> 否 Out -> 是
EBOOL	In -> 是 In-Out -> 否 Out -> 是	是

数据表变量之间的兼容性

	ARRAY[i..j) OF BOOL 目标	**ARRAY[i..j) OF EBOOL 目标**
ARRAY[i..j) OF BOOL 源	是	否
ARRAY[i..j) OF EBOOL 源	否	是

静态变量之间的兼容性

	BOOL (%MW:xi) **直接寻址**	**EBOOL (%Mi)** **直接寻址**
BOOL (Var:BOOL) 声明的变量	是	否
EBOOL (Var:EBOOL) 声明的变量	否	是

整数类型

内容预览

整数类型通过各种基数来表示一个数值。这些基数包括：

- 基数 10 (十进制)，它是缺省的基数。根据整数类型的不同，其数值可以是有符号的，也可以是无符号的。
- 基数 2 (二进制)。其数值是无符号的，带有前缀 **2#**
- 基数 8 (八进制)。其数值是无符号的，带有前缀 **8#**
- 基数 16 (十六进制)。其数值是无符号的，带有前缀 **16#**

注意：在十进制的演示中，如果所选的类型是有符号的，数值前面可以带有 + 或 - 号 (+ 号是可选的)。

整数类型 (INT)

具有 16 位格式的有符号类型。

下面的表格给出了每一种基数对应的取值范围：

基数	从 ...	到 ...
十进制 (有符号的)	-32768	32767
二进制 (无符号的)	2#0	2#1111111111111111
八进制 (无符号的)	8#0	8#177777
十六进制 (无符号的)	16#0	16#FFFF

双整数类型 (DINT)

具有 32 位格式的有符号类型。

下面的表格给出了每一种基数对应的取值范围：

基数	从 ...	到 ...
十进制 (有符号的)	-2147483648	2147483647
二进制 (无符号的)	2#0	2#11111111111111111111111111111111
八进制 (无符号的)	8#0	8#37777777777
十六进制 (无符号的)	16#0	16#FFFFFFFF

无符号的整数类型 (UINT)

具有 16 位格式的无符号类型。

下面的表格给出了每一种基数对应的取值范围：

基数	从 ...	到 ...
十进制	0	65535
二进制	2#0	2#1111111111111111
八进制	8#0	8#177777
十六进制	16#0	16#FFFF

无符号的双整数类型 (UDINT)

具有 32 位格式的无符号类型。

下面的表格给出了每一种基数对应的取值范围：

基数	从 ...	到 ...
十进制	0	4294967295
二进制	2#0	2#11111111111111111111111111111111
八进制	8#0	8#37777777777
十六进制	16#0	16#FFFFFFFF

时间类型

内容预览

时间类型 **T#** 或者 **TIME#** 由一个无符号的双整数 (UDINT) 类型 (参见*整数类型, 195 页*) 来表示。

它代表一个以毫秒来计算的时间，最多大约可以表示 49 天的时间。

可以表示数值的时间类型数据的单位包括：
- 天 **(D)**
- 小时 **(H)**
- 分钟 **(M)**
- 秒 **(S)**
- 毫秒 **(MS)**

输入一个数值

下面的表格针对允许的**时间**单位给出了输入时间类型最大数值的可能方式。

图解	**说明**
T#4294967295**MS**	以毫秒为单位的数值
T#4294967**S**_295**MS**	以秒 \ 毫秒为单位的数值
T#71582**M**_47**S**_295**MS**	以分钟 \ 秒 \ 毫秒为单位的数值
T#1193**H**_2**M**_47**S**_295**MS**	以小时 \ 分钟 \ 秒 \ 毫秒为单位的数值
T#49**D**_17**H**_2**M**_47**S**_295**MS**	以天 \ 小时 \ 分钟 \ 秒 \ 毫秒为单位的数值

8.2 BCD 格式的基本数据类型 (EDT)

内容预览

本节主题

本节描述了 BCD 格式 (二进制编码的十进制) 的数据类型，其中包括：

- 数据类型
- 日时间类型 (TOD)
- 日期和时间 (DT) 类型

本节内容

本节包含以下主题：

BCD 格式的数据类型概述

介绍

BCD 格式的数据类型属于 EDT (基本数据类型) 系列，该系列包含单一数据类型，而不包括导出数据类型 (数据表，结构，功能块)。

关于 BCD 格式的提示

二进制编码的十进制 (BCD) 格式通过四个位 (半字节) 来表示 **0 和 9 之间的**十进制数。

在这个格式中，用来对十进制数进行编码的四个位的组合会出现闲置的情况。

相关的表格：

十进制	二进制
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
	1010 (未使用)
	1011 (未使用)
	1100 (未使用)
	1101 (未使用)
	1110 (未使用)
	1111 (未使用)

用一个 16 位格式进行编码的例子：

十进制数值 2450	2	4	5	0
二进制数值	0010	0100	0101	0000

用一个 32 位格式进行编码的例子：

十进制数值 78993016	7	8	9	9	3	0	1	6
二进制数值	0111	1000	1001	1001	0011	0000	0001	0110

BCD 格式的数据类型

共有三种数据类型：

类型	名称	大小 (位)	缺省值
DATE	日期	32	D#1990-01-01
TIME_OF_DAY	日时间	32	TOD#00:00:00
DATE_AND_TIME	日期和时间	64	DT#1990-01-01-00:00:00

日期类型

内容预览

32 位格式编码的日期类型数据包含以下信息：

- 用一个 16 位栏 (4 个最高半字节) 编码的年
- 用一个 8 位栏 (2 个半字节) 编码的月
- 用一个 8 位栏 (2 个最低半字节) 编码的日

用 BCD 格式表示日期 2001-09-20：

年 (2001)	月 (09)	日 (20)
0010 0000 0000 0001	0000 1001	0010 0000

语法规则

日期类型数据通过以下方式输入：D#< 年 >-< 月 >-< 日 >

下面的表格显示了每个栏的上限 / 下限：

栏	限制	注释
年	[1990,2099]	
月	[01,12]	左面的 0 会一直显示出来，但是在输入时间的时候可以忽略
日	[01,31]	对 01\03\05\07\08\10\12 月
	[01,30]	对 04\06\09\11 月
	[01,29]	对 02 月 (闰年)
	[01,28]	对 02 月 (非闰年)

例子：

输入	注释
D#2001-1-1	月和日左面的 0 可以忽略
d#1990-02-02	前缀可以使用小写字母

日时间 (TOD) 类型

内容预览

32 位格式编码的日时间类型包含如下信息：

- 用一个 8 位栏 (2 个最高半字节) 编码的点钟
- 用一个 8 位栏 (2 个半字节) 编码的分
- 用一个 8 位栏 (2 个半字节) 编码的秒

注意：最低的 8 位未使用。

用 BCD 格式表示日时间 13：25：47：

点钟 (13)	分 (25)	秒 (47)	最低字节
0001 0011	0010 0101	0100 0111	未使用

语法规则

日时间类型数据按照以下方式输入：**TOD#**< 点钟 >：< 分 >：< 秒 >

下面的表格显示了每个栏的上限 / 下限：

栏	限制	注释
点钟	[00,23]	左面的 0 会一直显示出来，但是在输入时间的时候可以忽略
分	[00,59]	左面的 0 会一直显示出来，但是在输入时间的时候可以忽略
秒	[00,59]	左面的 0 会一直显示出来，但是在输入时间的时候可以忽略

例子：

输入	注释
TOD#1:59:0	点钟和秒左面的 0 可以忽略
tod#23:10:59	前缀可以使用小写字母
Tod#0:0:0	前缀可以是混合式的 (小写 \ 大写字母)

日期和时间 (DT) 类型

内容预览

用 64 位格式编码的**日期和时间**类型数据包含如下信息：
- 用一个 16 位栏 (4 个最高半字节) 编码的年
- 用一个 8 位栏 (2 个半字节) 编码的月
- 用一个 8 位栏 (2 个半字节) 编码的日
- 用一个 8 位栏 (2 个半字节) 编码的点钟
- 用一个 8 位栏 (2 个半字节) 编码的分
- 用一个 8 位栏 (2 个半字节) 编码的秒

注意：8 个最低位未使用。

例子：用 BCD 格式表示日期和时间 2000-09-20：13：25：47。

年 (2000)	月 (09)	日 (20)	点钟 (13)	分 (25)	秒 (47)	最低 字节
0010 0000 0000 0000	0000 1001	0010 0000	0001 0011	0010 0101	0100 0111	未使用

语法规则

日期和时间类型数据按照以下方式输入：
DT#< 年 >-< 月 >-< 日 >-< 点钟 >：< 分 >：< 秒 >

下面的表格显示了每个栏的上限 / 下限：

栏	限制	注释
年	[1990,2099]	
月	[01,12]	左面的 0 会一直显示出来，但是在输入时间的时候可以忽略
日	[01,31]	用于 01\03\05\07\08\10\12 月
	[01,30]	用于 04\06\09\11 月
	[01,29]	用于 02 月 (闰年)
	[01,28]	用于 02 月 (非闰年)
点钟	[00,23]	左面的 0 会一直显示出来，但是在输入时间的时候可以忽略
分	[00,59]	左面的 0 会一直显示出来，但是在输入时间的时候可以忽略
秒	[00,59]	左面的 0 会一直显示出来，但是在输入时间的时候可以忽略

例子：

输入	注释
DT#2000-1-10-0:40:0	月 \ 点钟 \ 秒左面的 0 可以忽略
dt#1999-12-31-23:59:59	前缀可以使用小写字母
Dt#1990-10-2-12:02:30	前缀可以是混合式的 (小写 \ 大写字母)

8.3 实数格式的基本数据类型 (EDT)

实数格式的数据类型概述

介绍

实数格式的数据类型属于 EDT (基本数据类型) 系列，该系列包含单一数据类型，而不包括导出数据类型 (数据表，结构，功能块)。

关于实数格式的提示

实数格式 (ANSI/IEEE 标准中的浮点) 以 32 位格式编码，相当于十进制的单精度浮点数。

表示浮点数值的 32 位被组织成 3 栏，它们是：
- **S**，符号位，可具有以下值
  - 0，表示正浮点数
  - 1，表示负浮点数
- **e**，用一个 8 位栏编码的指数
- **f**，用一个 23 位栏编码的定点部分

表示：

31	30 … 23	22 … 0
S	指数 **e**	定点部分 **f**

定点部分的值 (尾数) 在 [1， 0] 之间，通过以下公式来计算：

$$f = \sum_{i=0}^{22}(2^{i-23} \cdot b_i)$$

bi 代表对应于次序 (i=0 ～ 22) 的位值 (0 或 1)

可以表示的数的类型

以下数可以被表示出来：
- 标准的
- 非标准的
- 无穷大值
- 正 0 和负 0 值

下面的表格针对数的类型给出了各个栏内的数值：

e	f	S	数的类型
]0.255[	]0.1]	0 或 1	标准的
0	]0.1[	0 或 1	非标准的
255	0	0	正无穷大

e	f	S	数的类型
255	0	1	负无穷大
0	0	0	正 0
0	0	1	负 0

下面的表格给出了浮点数数值 V 的计算公式：

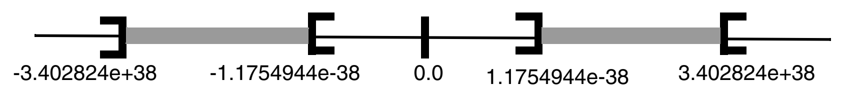
浮点数类型	数值 V
标准的	$(-1)^S \times 2^{(e-127)} \times (1+f)$
非标准的	$(-1)^S \times 2^{-126} \times f$

实数类型

介绍：

类型	大小 (位)	缺省值
REAL	32	0.0

数值范围 (灰色的部分)：



如果一个计算结果：

- 在 -1.175494e-38 和 1.175494e-38 之间，它会被近似为 0
- 小于 -3.402824e+38，会显示 `-INF` 符号 (负无穷)
- 大于 +3.402824e+38，会显示 `INF` 符号 (正无穷)
- 未定义 (负数的平方根)，会显示 `NAN` 符号

例子：

浮点数 -5.934113e-18 的表示。

S 符号	指数 e	定点部分 f
1	10010010	10110101000110000100001

等价输入的例子：

输入 ...	等价于 ...	以及 ...
+0.456	0.456	.456
-1.32e12	-132E10	-.132e+13
1.0E+6	1000000.	1.e6

8.4 字符串格式的基本数据类型 (EDT)

字符串格式数据类型概述

介绍

字符串格式的数据类型属于 EDT (基本数据类型) 系列，该系列包含单一数据类型，而不包括导出数据类型 (数据表，结构，功能块)。

字符串类型

字符串格式用来表示一个 ASCII 字符串，其中的每一个字符都用 8 位格式编码。

字符串类型数据的特性如下：
- 缺省情况下，一个字符串内含有 16 个字符 (不包括字符串尾)
- 字符串由 16#20 和 16#FF 之间的 ASCII 字符组成 (十六进制表示)
- 如果字符串是空的，那么字符串尾 (ASCII 代码 “ZERO”) 是字符串的第一个字符
- 字符串的最大尺寸是 65534 个字符

字符串的尺寸可以在使用 **STRING[<size>]** 命令进行类型定义的时候进行优化，借助无符号整数 UINT <size> 定义 1 到 65534 ASCII 字符之间的字符串。

语法规则

输入的两端带有单引号字符 "’ " (ASCII 代码为 16#27)。

$ (美元) 符号是一个特殊字符，它后面会带有特定的字母，这些字母表示：
- $L 或者 $l，到下一行 (进入行)
- $N 或者 $n，到下一行的起始处 (新的一行)
- $P 或者 $p，到下一页
- $R 或者 $r，回车换行
- $T 或者 $t　制表 (Tab)
- $$，代表字符串中的 $ 字符
- $'，代表字符串中的单引号字符

例子

输入的例子：

类型	输入	字符串的内容 • 表示字符串尾 * 表示空字节
STRING	‘ABCD’	ABDC•************ （16 个字符）
STRING[4]	‘john’	john•
STRING[10]	‘It$’s john’	It’ john•*
STRING[5]	”	•*****
STRING[5]	’$”	’•****
STRING[5]	‘the number’	the no•
STRING[13]	’0123456789’	0123456789•***
STRING[5]	‘RL’	<cr><lf>•***
STRING[5]	’$$1.00’	$1.00•

8.5 位串格式的基本数据类型 (EDT)

内容预览

本节主题

本节描述了位串格式的数据类型，其中包括：
- 字节类型
- 字类型
- 双字类型

本节内容

本节包含以下主题：

位串格式的数据类型概述

介绍

位串格式的数据类型属于 EDT (基本数据类型) 系列，该系列包含单一数据类型，而不包括导出数据类型 (数据表，结构，功能块)。

关于位串格式的提示

这种格式的特点在于它所有的组成位都不表示数值，而是一个独立位的组合。

属于这种格式类型的数据可以用三种基数表示。这些基数是：
- 十六进制 (16#)
- 八进制 (8#)
- 二进制 (2#)

位串格式的数据类型

有三种数据类型：

类型	大小 (位)	缺省值
BYTE	8	0
WORD	16	0
DWORD	32	0

位串类型

字节类型

字节类型用 8 位格式进行编码。
下面的表格给出了每一种可用基数对应的下限 / 上限：

基数	下限	上限
十六进制	16#0	16#FF
八进制	8#0	8#777
二进制	2#0	2#11111111

演示实例：

数据内容	用一种基数给出的演示
00001000	16#8
00110011	8#63
00110011	2#110011

字类型

字类型用 16 位格式进行编码。
下面的表格给出了每一种可用基数对应的下限 / 上限：

基数	下限	上限
十六进制	16#0	16#FFFF
八进制	8#0	8#177777
二进制	2#0	2#1111111111111111

演示实例：

数据内容	用一种基数给出的演示
0000000011010011	16#D3
1010101010101010	8#125252
0000000011010011	2#11010011

双字类型

双字类型用 32 位格式进行编码。

下面的表格给出了每一种可用基数对应的下限 / 上限：

基数	下限	上限
十六进制	16#0	16#FFFFFFFF
八进制	8#0	8#37777777777
二进制	2#0	2#11111111111111111111111111111111

演示实例：

数据内容	用一种基数给出的演示
00000000000010101101110011011110	16#ADCDE
00000000000000010000000000000000	8#200000
00000000000010101011110011011110	2#10101011110011011110

8.6 导出数据类型 (DDT/IODDT)

内容预览

本节主题

本节介绍导出数据类型，其中包括：
- 数据表 (DDT)
- 结构
  - 关于输入 / 输出数据 (IODDT) 的结构
  - 关于其他数据 (DDT) 的结构

本节内容

本节包含以下主题：

数据表

什么是数据表？

数据表是包含一系列同类数据的数据项，比如：

- 基本数据 (EDT)
  - 一组 BOOL 字
  - 一组 UINT 整数字
  - 其他
- 导出数据 (DDT)
  - 一组 WORD 数据表
  - 一组结构
  - 其他

特性

数据表的特性通过两个参数体现出来：

- 一个定义它的组织形式的参数 (数据表维数)
- 一个定义它包含的数据类型的参数

注意：最复杂的数据表有六维。

包含这两个参数的语法如下所示：

ARRAY [< 数据表的维数 >**]** OF < 数据类型 >

< 1 维 >， < 2 维 >， < n 维 >

< 最小范围 >..< 最大范围 >

< 最小范围 > **必须小于** < 最大范围 >

对数据表进行定义和实例化

定义数据表类型：

```
X: ARRAY[1..10] OF BOOL
```

对数据表进行实例化

```
Tab_1: X
Tab_2: ARRAY[1..10] OF BOOL
```

实例 Tab_1 和 Tab_2 是同样的类型，具有相同的维数，它们唯一的区别就是在实例化的过程中：

- Tab_1 类型采用 X 名称，
- Tab_2 类型必须被定义 (未命名的数据表)。

注意：对类型进行命名会给使用者带来方便，因为在这种情况下，只需要一次操作就可以完成所需的修改。如果没有命名，那么有每一个实例都要进行一次修改。

例子

下面的表格各种维数的数据表的实例：

输入	注释
Tab_1：ARRAY[1..2] OF BOOL	带有 2 个布尔字的 1 维数据表
Tab_2：ARRAY[-10..20] OF COORD	带有 31 个 COORD 类结构 (由用户定义的结构) 的 1 维数据表
Tab_3：ARRAY[1..10, 1..20] OF INT	带有 10x20 个整数的 2 维数据表
Tab_4：ARRAY[0..2, -1..1, 201..300, 0..1] OF REAL	带有 3x3x100x2 个实数的 4 维数据表

访问数据表 Tab_1 和 Tab_3 的一个数据项：

```
Tab_1[2]
;To access second element

Tab_3[4][18]
;To access eighteenth element of the fourth sub-table
```

数据表间的分配规则

有如下 4 个数据表：

```
Tab_1:ARRAY[1..10] OF INT
Tab_2:ARRAY[1..10] OF INT
Tab_3:ARRAY[1..11] OF INT
Tab_4:ARRAY[101..110] OF INT


Tab_1:=Tab_2; Assignment authorized
Tab_1:=Tab_3; Assignment refused (non-IEC compliant)
Tab_1:=Tab_4; Assignment refused (non-IEC compliant)
```

结构

什么是结构？

是包含一系列异类数据的数据项，比如：

- 一组 BOOL，字， UINT，等等 (EDT 结构)
- 一组数据表 (DDT 结构)
- 一组实数，双字，数据表，等等 (EDT 和 DDT 结构)

特性

组成结构的数据具有以下内容：

- 一个类型
- 一个用来识别该数据的名称
- 一个描述该数据的作用的注释 (可选)

结构类型定义：

```
IDENT
    Surname: STRING[12]
    First name: STRING[16]
    Age: UINT

;The IDENT type structure contains a UINT type data item and two
STRING type data
```

一个 IDENT 类型结构的两个数据实例定义：

```
Person_1: IDENT
Person_2: IDENT

;The 实例 Person_1 and Person_2 are of IDENT Structure type
```

访问结构的数据

Person_1 IDENT 类型实例的数据：

```
Person_1.Name  ;To access name of Person_1

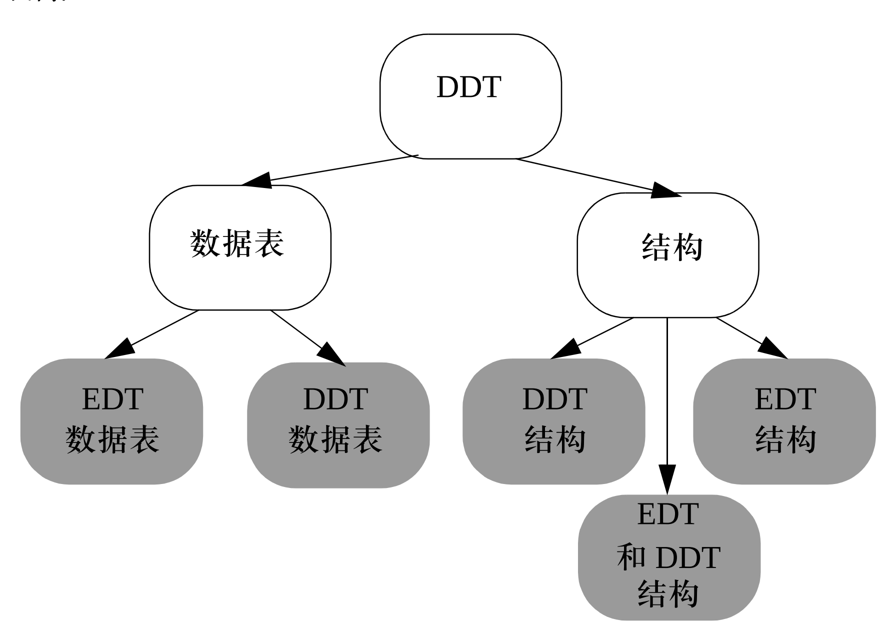
Person_1.Age    ;To access age of Person_1
```

导出数据类型系列 (DDT) 概述

介绍

DDT(导出数据类型) 系列包括 “导出”数据类型，比如：
- 数据表
- 结构

图例：



特性

属于 DDT 系列的数据项由以下内容组成：
- 由用户定义的类型名称(参见*类型\实例名称的语法规则，187 页*) (最多为32个字符) (对数据表来说不是必需的，但是推荐用户采用) (参见*对数据表进行定义和实例化，214 页*)
- 类型 (结构或者数据表)
- 一个可选的注释 (最多为 1024 个字符)。可用的字符对应 ASCII 码的 32 到 255
- 这些元素的描述 (结构)
  - 元素名称 (参见*类型 \ 实例名称的语法规则，187 页*) (最多为 32 个字符)
  - 元素类型
  - 一个可选的注释 (最多为 1024 个字符)，用来描述它的作用。可用的字符对应 ASCII 码的 32 到 255

- 以下等方面的信息：
- 类型版本号
- 上一次修改代码或者内部变量或接口变量的日期
- 一个可选的描述文件 (32767 个字符)，用来描述块功能和相关的各种修改

注意：数据表或者结构的总尺寸不超过 64 K 字节。

例子

类型定义

```
COORD
    X: INT
    Y: INT
;COORD type structure

SEGMENT
    Origin: COORD
    Destination: COORD
;SEGMENT type structure containing 2 COORD type structures

OUTLINE: ARRAY[0..99] OF SEGMENT
;OUTLINE type table containing 100 SEGMENT type structures

DRAW
    Color: INT
    Anchor: COORD
    Pattern: ARRAY[0..15,0..15] OF WORD
    Contour: OUTLINE
;DRAW type structure
```

访问 DRAW 类型结构实例的数据

```
Cartoon: DRAW
;Instance of DRAW type structure

Cartoon.Pattern[15,15]
;Access to last data item in the Pattern table of the Cartoon
structure

Cartoon.Contour[0].Origin.X
;Access to data item X of the COORD structure belonging to the
first SEGMENT structure of the Contour table.
```

DDT：内存占用

内容预览

DDT 存储在 PLC 的内存中，它的存储顺序与其元素的声明顺序一致。
不过，下面的规则同样也适用。

原则

存储原则如下：

- 元素按照它们在结构中被声明的顺序进行存储，
- 基本元素是字节 (在内存字节上的数据对齐)，
- 每一个元素都有一个对齐规则：
  - BOOL 和 BYTE 类型在奇数字节或者偶数字节都要进行对齐，
  - 所有其他基本类型都在偶数字节进行对齐，
  - 结构和数据表如果只包含BOOL和BYTE元素，那么它们要根据BOOL和BYTE类型的对齐规则进行对齐，否则它们在内存的偶数字节进行对齐。



警告

请留意从 Concept 转换的应用程序的兼容性。

在 **Concept** 应用程序中，数据结构不会处理偏移量的任何移位 (在内存中，所有类型的元素都是依次排列)。所以，我们建议您检查一切相关内容，尤其是在使用定位于 “状态 RAM” (存在着移位的风险) 的 DDT 或者用来和其他设备进行通信的功能 (传输所用尺寸与 Concept 程序不同) 时，要检查数据的一致性。

如果未能遵守此处的警告，可能会造成伤害或者设备损坏。

例 1

下面的表格给出了数据结构的一些例子：

偏移	结构描述	
	Para_PWM1	
偏移 0		t_period: TIME
偏移 4		t_min: TIME
偏移 8		in_max: REAL
	Para_PWM1 类型结构	
	Mode_TOTALIZER	
偏移 0		hold: BOOL
偏移 1		rst: BOOL
	Mode_TOTALIZER 类型结构	
	Info_TOTALIZER	
偏移 0		outc: REAL
偏移 4		cter: UINT
偏移 6		done: BOOL
偏移 7		**保留给对齐功能**
	Info_TOTALIZER 类型结构	

例 2

下面的表格给出了两个带有数据表的数据结构：

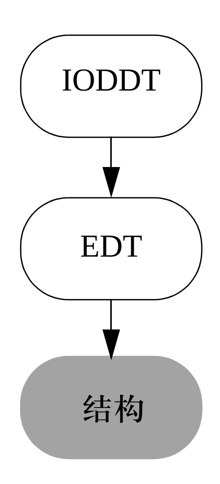
偏移	结构描述
	EHC105_Out
偏移 0	Quit: BYTE
偏移 1	Control: ARRAY [1..5] OF BYTE
偏移 6	Final: ARRAY [1..5] OF DINT
	EHC105_Out 类型结构
	CPCfg_ex
偏移 0	Profile_type: INT
偏移 2	Interp_type: INT
偏移 4	Nb_of_coords: INT
偏移 6	Nb_of_points: INT
偏移 8	reserved: ARRAY [0.0.4] OF BYTE
偏移 13	**保留给 Master_offset 偶数字节的对齐功能**
偏移 14	Master_offset: INT
偏移 16	Follower_offset: INT
	CPCfg_ex 类型结构

输入 / 输出导出数据类型 (IODDT) 概述

内容预览

IODDT (输入输出导出数据类型) **由厂商定义**，它包含了属于应用程序专用模块通道的 EDT 系列语言对象。

图例：



IODDT 类型结构的尺寸 (组成它们的元素的数量) 取决于它们所表示的通道或者输入 \ 输出模块。

一个特定的输入 \ 输出模块可以拥有多个 IODDT。

它与常规结构的不同之处在于：
- IODDT 结构由厂商预先定义好
- 组成 IODDT 结构的元素没有连续的内存分配，但是在模块中拥有一个明确的地址

例子

用于一个模拟模块输入 \ 输出通道的 IODDT 结构

```
ANA_IN_GEN    ;ANA_IN_GEN type structure
   Value:INT ;Input value
   Err:  BOOL   ;Channel error
```

访问 ANA_IN_GEN 类型的实例数据：

```
 Cistern_Level: ANA_IN_GEN
; ANA_IN_GEN type instance which corresponds for example
to a tank level sensor

Cistern_Level.Value ;Reading of the channel input value
Cistern_Level.Err ;Reading of channel error bit
```

通过直接寻址进行访问：

```
For channel 0 of module 2 of rack 0 we obtain:

Cistern_Level       corresponds to %CH0.2.0
Cistern_Level.Value corresponds to %IW0.2.0.0
Cistern_Level_Err   corresponds to %I0.2.0.ERR
```

8.7 功能块数据类型 (DFB\EFB)

内容预览

本节主题

本节描述了功能块数据类型，其中包括：
- 用户功能块 (DFB)
- 基本功能块 (EFB)

本节内容

本节包含以下主题：

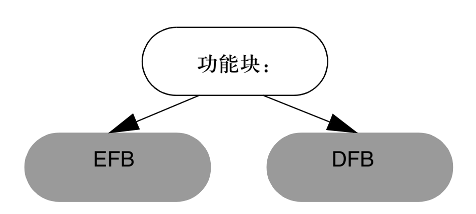
功能块数据类型系列概述

介绍

功能块数据类型系列包括：

- 基本功能块 (EFB) (参见*数据类型系列综述，183 页*) 类型系列
- 用户功能块 (DFB) (参见*数据类型系列综述，183 页*) 类型系列

图例：

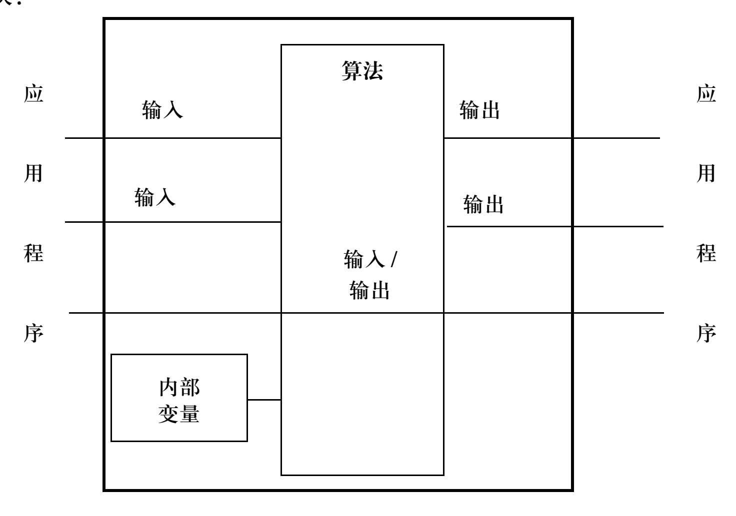


功能块是包含如下内容的实体：

- 充当应用程序接口的输入和输出变量
- 一个对输入变量进行算法处理并完成输出变量的操作
- 通过处理算法对私有和公共内部变量进行操作

图例

功能块：



用户功能块 (DFB)

用户功能块类型 (导出功能块) 由用户通过一种或者多种语言进行开发 (根据代码段的数量)。这些语言包括：
- 梯形图语言
- 结构化文本语言
- 指令表语言
- 功能块语言 FBD

DFB 类型可以具有一个或者多个实例，每个实例通过一个名称 (符号) 来引用，并拥有 DFB 数据类型。

基本功能块 (EFB)

基本功能块 (EFB) 由厂商提供，用 C 语言编写。

用户可以创建自己的 EFB，在创建过程中要用到一个可选的软件工具 “**SDKC**”。

EFB 类型可以具有一个或者多个实例，每个实例通过一个名称 (符号) 来引用，并拥有 EFB 数据类型。

功能块数据类型 (EFB\DFB) 的特性

类型定义

EFB 或者 DFB 功能块类型通过如下内容进行定义：

- 类型名称 (参见*类型 \ 实例名称的语法规则，187 页*)，由用户为 DFB 进行定义，
- 一个可选的注释。可用的字符串对应 ASCII 码的 32 到 255，
- 应用程序接口数据：
  - 输入，在读 \ 写模式下不能对应用程序进行访问，但是可以被功能块代码读入，
  - 输入 \ 输出，在读 \ 写模式下不能对应用程序进行访问，但是可以被功能块代码读入和写出，
  - 输出，在只读模式下可以对应用程序进行访问，可以被功能块代码读入和写出。
- 内部数据：
  - 公共内部数据，在读 \ 写模式下可以对应用程序进行访问，可以被功能块代码读入和写出，
  - 私有内部数据，不能对应用程序进行访问，但是可以被功能块代码读入和写出。
- 代码：
  - 对于 DFB，用户用 PLC 语言 (结构化文本，指令表，梯形图语言，功能块语言) 编写，如果 IEC 选项有效，它可以在一个单一的代码段内进行结构化处理；如果该选项无效，它可以在若干个代码段中进行结构化处理
  - 对于 EFB，它用 C 语言编写。
- 如下信息：
  - 类型版本号，
  - 代码，内部变量或者接口变量上一次修改的日期。
  - 一个可选的描述文件 (32767 个字符)，描述功能块和与之相关的各种修改。

特性

下面的表格给出了构成一个类型的元素的特性：

元素	**EFB**	**DFB**
名称	32 个字符	32 个字符
注释	1024 个字符	1024 个字符
输入数据	最多 32 个	最多 32 个
输入 / 输出数据	最多 32 个	最多 32 个
输出数据	最多 32 个	最多 32 个
接口的数量 (输入 + 输出 + 输入 / 输出)	最多 32 个 **(2)**	最多 32 个 **(2)**
公共数据	没有限制 **(1)**	没有限制 **(1)**
私有数据	没有限制 **(1)**	没有限制 **(1)**
编程语言	C 语言	语言： • 结构化文本， • 指令表， • 梯形图语言， • 功能块。
代码段		代码段通过以下内容进行定义： • 一个名称 (最多 32 个字符)， • 一个确认条件， • 一个注释 (最多 256 个字符)， • 一个保护： • 没有保护， • 只读， • 读 \ 写模式。 代码段不能访问在应用程序中声明的变量，以下内容是例外： • 系统双字 %SDi， • 系统字 %SWi， • 系统位 %Si。

(1)：唯一的限制是 PLC 内存的大小。

(2)：EN 输入和 ENO 输出不予考虑。

属于功能块的元素的特性

什么是元素 ?

- 每一个元素 (接口数据或者内部数据) 都通过以下内容进行定义：
- 一个名称 (参见*类型 \ 实例名称的语法规则，187 页*) (最多 32 个字符)，由用户定义，
- 一个类型，可以隶属于以下系列：
- 基本数据类型 (EDT)，
- 导出数据类型 (DDT)，
- 功能块数据类型 (EFB\DFB)。
- 一个可选的注释 (最多 1024 个字符)。可用的字符串对应 ASCII 码的 32 到 255，
- 一个初始值，
- 一个从应用程序进行访问的权限 (属于 DFB 的应用程序代码段或者应用程序 参见 "功能块类型定义 (接口和内部变量)" (参见*类型定义，227 页*)，
- 一个从通信请求进行访问的权限，
- 一个公共变量备份标记。

属于 DFB 的元素的可用数据类型

可用数据类型包括：

DFB 元素	**EDT** 类型	**DDT** 类型				**ANY...**	**功能块**类型
		IODDT	未命名的数据表	ANY_ARRAY	其他		
输入数据	是	无	是	是	是	是 **(2)**	无
输入 / 输出数据	是 **(1)**	是	是	是	是	是 **(2)**	无
输出数据	是	无	是	无	是	是 **(2) (3)**	无
公共数据	是	无	是	无	是	无	无
私有数据	是	无	是	无	是	无	是

(1)：对于在 Quantum PLC 上使用的 EBOOL 类静态数据不可用
(2)：对于 BOOL 和 EBOOL 类数据不可用
(3)：必须在 DFB 的执行期间完成，在 DFB 以外不可用

属于 EFB 的元素的可用数据类型

可用的数据类型包括：

EFB 元素	EDT 类型	DDT 类型				ANY...	功能块类型
		IODDT	未命名的数据表	ANY_ARRAY	其他		
输入数据	是	无	无	是	是	是 **(1)**	无
输入 / 输出数据	是	是	无	是	是	是 **(1)**	无
输出数据	是	无	无	无	是	是 **(1) (2)**	无
公共数据	是	无	无	无	是	无	无
私有数据	是	无	无	无	是	无	是

(1)：对 BOOL 和 EBOOL 类数据不可用

(2)：必须在 EFB 的执行期间完成，在 EFB 以外不可用

属于 DFB 的元素的初始值

下面的表格说明了是否能够从 DFB 类型定义或者 DFB 实例中输入初始值：

DFB 元素	从 DFB 类型	从 DFB 实例
输入数据 (无 ANY... 类型)	是	是
输入数据 (ANY... 类型)	否	否
输入 / 输出数据	否	否
输出数据 (无 ANY... 类型)	是	是
输出数据 (ANY... 类型)	否	否
公共数据	是	是
私有数据	是	否

属于 EFB 的元素的初始值

下面的表格说明了是否能够从 EFB 类型定义或者 EFB 实例中输入初始值：

EFB 元素	**从 EFB 类型**	**从 EFB 实例**
输入数据 (没有 ANY... 类型参见*泛型数据类型概述，232 页*)	是	是
输入数据 (ANY... 类型)	否	否
输入 / 输出数据	否	否
输出数据 (ANY... 类型)	是	是
输出数据 (ANY... 类型)	否	否
公共数据	是	是
私有数据	是	否

8.8 泛型数据类型 (GDT)

泛型数据类型概述

介绍

泛型数据类型是按照一定层次结构组织起来的常规数据类型 (EDT，DDT) 组，它们能够专门确定这些常规数据类型组之间的兼容性。

这些数据类型组通过前缀 **ANY_xxx** 来识别，不过这些前缀在任何情况下都不能用于数据实例。

普通数据类型被应用于以下方面：**功能块 (EFB\DFB) 和基本功能 (EF) 数据类型系列**，其作用是定义哪些数据类型能够和它们与如下各项的接口兼容：

- 输入，
- 输入 / 输出，
- 输出。

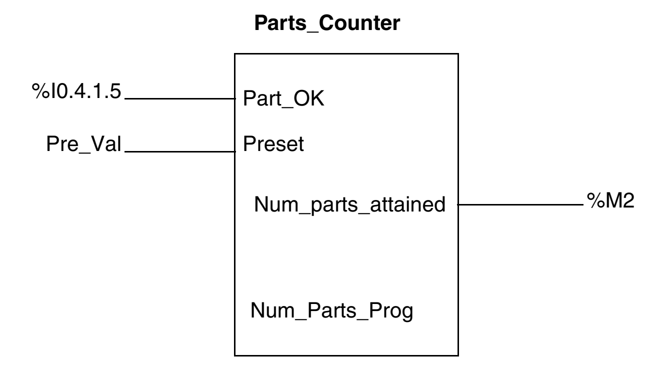
内容预览

泛型数据类型的层次结构介绍

- **ANY**
  - **ANY_ELEMENTARY**
    - **ANY_MAGNITUDE_OR_BIT**
      - **ANY_MAGNITUDE**
        - **ANY_NUM**
          - **ANY_REAL**
            - **REAL**
          - **ANY_INT**
            - **DINT, INT, UDINT, UINT**
        - **TIME**
      - **ANY_BIT**
        - **DWORD, WORD, BYTE, BOOL**
    - **ANY_STRING**
      - **STRING**
    - **ANY_DATE**
      - **DATE_AND_TIME, DATE, TIME_OF_DAY**
    - **EBOOL**
  - **ANY_DERIVED**
    - **ANY_ARRAY**
      - **ANY_ARRAY_ANY_EDT**
        - **ANY_ARRAY_ANY_MAGNITUDE**
          - **ANY_ARRAY_ANY_NUM**
            - **ANY_ARRAY_ANY_REAL**
              - **ANY_ARRAY_REAL**
            - **ANY_ARRAY_ANY_INT**
              - **ANY_ARRAY_DINT**
              - **ANY_ARRAY_INT**
              - **ANY_ARRAY_UDINT**
              - **ANNY_ARRAY_UINT**
          - **ANY_ARRAY_TIME**
        - **ANY_ARRAY_ANY_BIT**
          - **ANY_ARRAY_DWORD**
          - **ANY_ARRAY_WORD**
          - **ANY_ARRAY_BYTE**
          - **ANY_ARRAY_BOOL**
        - **ANY_ARRAY_ANY_STRING**
          - **ANY_ARRAY_STRING**
        - **ANY_ARRAY_ANY_DATE**
          - **ANY_ARRAY_DATE_AND_TIME**
          - **ANY_ARRAY_DATE**
          - **ANY_ARRAY_TIME_OF_DAY**
        - **ANY_ARRAY_EBOOL**
      - **ANY_ARRAY_ANY_DDT**
    - **ANY_STRUCTURE**
      - **ANY_DDT**
      - **ANY_IODDT**
      - **ANY_FFB**
        - **ANY_EFB**
        - **ANY_DFB**

例子

以如下 DFB 为例：



Preset 输入参数可以被定义为 ANY_INT 类型，这表示 **Pre_Val** 变量可以是 INT， UINT， DINT 或者 UDINT 类型。

8.9 属于顺序功能图 (SFC) 的数据类型

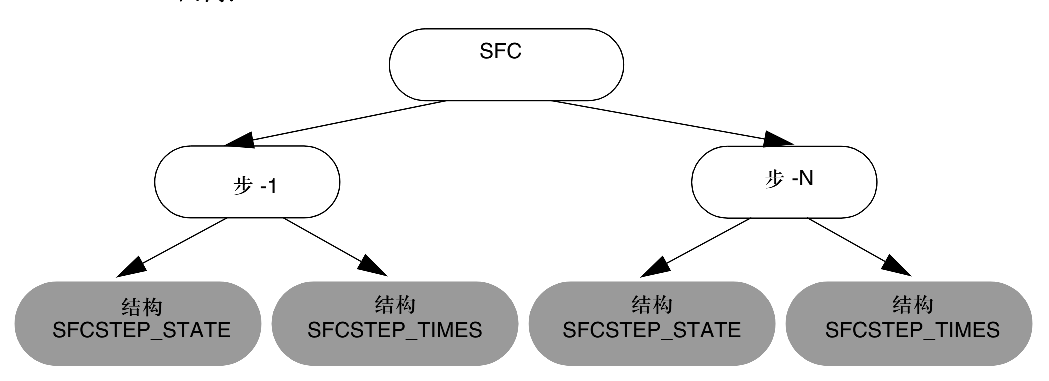
顺序功能图系列的数据类型概述

介绍

顺序功能图 (SFC) 数据类型系列包括**导出**数据类型，比如恢复图表属性和状态及其组件行为的结构。

每一个步都用两个结构来表示，亦即：
- **SFCSTEP_STATE** 结构，
- **SFCSTEP_TIMES** 结构。

图例：



> **注意**：两个结构类型 **SFCSTEP_STATE** 和 **SFCSTEP_TIMES** 也都与顺序功能图的每一个宏步相链接。

SFCSTEP_STATE 结构类型定义

该结构包括与步或者宏步状态相链接的所有数据类型。

这些数据类型包括：
- **x**：BOOL 基本数据类型 (EDT)，在步处于有效状态时，它就为 TRUE 值，
- **t**：TIME 基本数据类型 (EDT)，包含步的活动时间。当它处于无效状态时，步会保持其数值不变，直到再次被激活为止，
- **tminErr**：BOOL 基本数据类型 (EDT)，如果步的活动时间小于程序设定的最小活动时间，它就为 TRUE 值，
- **tmaxErr**：BOOL 基本数据类型 (EDT) ，如果步的活动时间大于程序设定的最大活动时间，它就为 TRUE 值，

这些数据类型可以在只读模式下被应用程序访问。

SFCSTEP_TIMES 结构类型定义

这个结构包括与步或者宏步的运行参数定义相链接的所有数据类型。

这些数据类型包括：
- **delay**：TIME 基本数据类型 (EDT)，定义位于有效步下游的转换的查询延迟时间，
- **tmin**：TIME 基本数据类型 (EDT)，包含一个最小数值，在该数值范围内，步必须被执行。如果未能做到这一点，数据 tmin.Err 会变为 TRUE，
- **tmax**：TIME 基本数据类型 (EDT)，包含一个最大数值，在该数值范围内，步必须被执行。如果未能做到这一点，数据 tmax.Err 会变为 TRUE。

这些数据类型只能从 SFC 编辑器进行访问。

SFCSTEP_STATE 结构的数据访问语法

这个结构的实例名称与顺序功能图的步或者宏步相对应

语法	注释
Name_Step.x	用来查看步的状态 (有效 \ 无效)
Name_Step.t	用来发查看步当前或者总有效时间
Name_Step.tminErr	用来查看步的最小有效时间是否小于在 Name_Step.tmin 中所设定的时间
Name_Step.tmaxErr	用来查看步的最大有效时间是否大于在 Name_Step.tmax 中所设定的时间

8.10 数据类型之间的兼容性

数据类型之间的兼容性

介绍

本节介绍了以下各个系列内部的类型兼容性的各种规则：
- 基本数据类型 (EDT) 系列，
- 导出数据类型 (DDT) 系列，
- 泛型数据类型 (GDT) 系列。

基本数据类型 (EDT) 系列

基本数据类型 (EDT) 系列包含以下子系列：
- 二进制格式数据类型子系列，
- BCD 格式数据类型子系列，
- 实数格式数据类型子系列，
- 字符串格式数据类型子系列，
- 位串格式数据类型子系列，

在两种不同的数据类型之间不存在任何兼容性，即便它们属于同一个子系列也是如此。

导出数据类型 (DDT) 系列

导出数据类型 (DDT) 系列包含以下子系列：

- 数据表类型子系列，
- 结构类型子系列：
  - 关于输入 / 输出数据 (IODDT) 的结构，
  - 关于其他数据的结构。

关于结构的规则：
两种结构的元素如果满足以下条件，那么这两种结构是兼容的：

- 名称相同，
- 类型相同，
- 组织顺序相同。

有四种结构：

```
ELEMENT_1
  My_Element: INT
  Other_Element: BOOL
;ELEMENT_1 type structure


ELEMENT_2
  My_Element: INT
  Other_Element: BOOL
;ELEMENT_2 type structure

ELEMENT_3
  Element: INT
  Other_Element: BOOL
;ELEMENT_3 type structure

ELEMENT_4
  Other_Element: BOOL
  My_Element: INT
;ELEMENT_4 type structure
```

结构类型之间的兼容性

类型	ELEMENT_1	ELEMENT_2	ELEMENT_3	ELEMENT_4
ELEMENT_1		是	否	否
ELEMENT_2	是		否	否
ELEMENT_3	否	否		否
ELEMENT_4	否	否	否	

关于数据表的规则

两个数据表如果满足如下条件，那么它们就是兼容的：

- 它们的维数和维的顺序一致，
- 每一个相应的维都是同样的类型。

有五种数据表：

```
TAB_1: ARRAY[10..20]OF INT
;Table one dimension of TAB_1 type

TAB_2: ARRAY[20..30]OF INT
;Table one dimension of TAB_2 type

TAB_3: ARRAY[20..30]OF INT
;Table one dimension of TAB_3 type

TAB_4: ARRAY[20..30]OF TAB_1
;Table one dimension of TAB_4 type

TAB_5: ARRAY[20..30,10..20]OF INT
;Table two dimensions of type TAB_5
```

数据表类型之间的兼容性：

类型 ...	和类型 ...	是 ...
TAB_1	TAB_2	不兼容的
TAB_2	TAB_3	兼容的
TAB_4	TAB_5	兼容的
TAB_4[25]	TAB_5[28]	兼容的

泛型数据类型 (GDT) 系列

泛型数据类型 (GDT) 系列是由一些组构成，这些组按照一定的层次结构组织起来，它们包含属于如下系列的数据类型：

- 基本数据类型 (EDT)，
- 导出数据类型 (DDT)。

规则：

如果一个泛型数据类型与一个常规数据类型在层次结构上相关联，那么它们就是兼容的 (参见 “泛型数据类型” 一节) (参见*内容预览，233 页*)。

如果一个泛型数据类型与一个普通数据类型在层次结构上相关联，那么它们就是兼容的 (参见 “泛型数据类型” 一节) (参见*内容预览，233 页*)。

例子：

```
The INT type is compatible with the ANY_INT or ANY_NUM or
ANY_MAGNITUDE types.

The INT type is not compatible with the ANY_BIT or ANY_REAL types.

The ANY_INT generic type is compatible with the ANY_NUM type.

The ANY_INT generic type is not compatible with the ANY_REAL type.
```

数据实例

9

内容预览

本章内容

本章描述了数据实例及其特性。这些数据实例包括：

- 非定位数据实例，
- 定位型数据实例，
- 直接寻址数据实例。

本章内容

本章包含以下内容：

数据类型实例

介绍

什么是数据类型实例？(参见*数据实例概述，185 页*)

数据类型实例通过以下方式引用：
- **一个名称 (符号)**，在这种情况下，我们将数据称为**非定位型的**，因为它没有定义内存地址，但是系统会自动进行内存分配，
- 由厂商定义的**一个名称 (符号)** 和**一个拓扑地址**，在这种情况下，我们将数据称为**定位型的**，因为它有已知的内存地址，
- 由厂商定义的**一个拓扑地址**，在这种情况下，我们将数据称为**直接寻址**，它有已知的内存地址。

非定位数据实例

非定位数据实例由 PLC 操作系统进行管理，**用户不知道它们**在内存中的物理位置。

非定位数据实例通过属于以下系列之一的数据类型进行定义：
- 基本数据类型 (EDT)，
- 导出数据类型 (DDT)，
- 功能块数据类型 (EFB\DFB)。

例子：

```
Var_1: BOOL
;Instance of EDT family of Boolean type with 1 byte memory allocation

Var_2: UDINT
;Instance of EDT family of double unsigned integer type with 4 byte
memory allocation

Var_3: ARRAY[1..10]OF INT
;Instance of DDT family of table type with 20 byte memory allocation

COORD
   X: INT
   Y: INT
Var_4: COORD
;Instance of DDT family of COORD type structure with 4 byte memory
allocation
```

定位数据实例

定位数据实例在 PLC 中有一个预先定义的内存位置，用户知道该位置。

定位数据实例通过属于以下系列之一的数据类型进行定义：
- 基本数据类型 (EDT)，
- 导出数据类型 (DDT)，
- 顺序功能图数据类型 (SFC)。

例子：

```
Var_1: EBOOL AT %M100
;Instance of EDT family of Boolean type (with 1 byte memory
allocation) predefined in %M100

Var_2: BOOL AT %I2.1.0.ERR
;Instance of EDT family of Boolean type (with 1 byte memory
allocation) predefined in %I2.1.0.ERR

Var_3: INT AT %MW10
;Instance of EDT family of integer type (with 2 byte memory
allocation) predefined in %MW10

Var_4: INT AT %MW10
;Prohibited as Var_3 has the same memory allocation and is of the
same type

Var_5: WORD AT %MW10
;Instance of EDT family of WORD type (with 2 byte memory allocation)
predefined in %MW10

Var_6: ARRAY[1..10]OF INT AT %MW50
;Instance of EDT family of table type (with 20 byte memory
allocation) predefined from %MW50

COORD
   X: INT
   Y: INT

Var_7: COORD AT %MW20
;Instance of DDT family of COORD structure type (with 4 byte memory
allocation) predefined from %MW20
```

注意：顺序功能图 (SFC) 数据类型实例在插入到应用程序的时候被创建，它有一个缺省的名称，用户可以修改这个名称。

直接寻址数据实例

直接寻址数据实例在 PLC 内存或者应用程序专用模块内有一个预先定义的位置，**用户知道该位置**

直接寻址数据实例通过属于基本数据类型 (EDT) 系列的类型进行定义

直接寻址数据实例的例子：

内部	常数	系统	输入 / 输出	网络
%Mi		%Si	%Q, %I	
%MWi	%KWi	%SWi	%QW, %IW	%NW
%MDi	%KDi	%SDi	%QD, %ID	
%MFi	%KFi			

数据实例属性

内容预览

数据实例的属性是定义的信息，这些信息包括：
- 它的名称(参见*类型\实例名称语法规则，187 页*) (除了直接寻址数据实例(参见*直接寻址数据实例，247 页*))，
- 它的拓扑地址 (除了非定位数据类型实例)，
- 它的数据类型，可以属于以下系列中的一种：
  - 基本数据类型 (EDT)，
  - 导出数据类型 (DDT)，
  - 功能块数据类型 (EFB\DFB)，
  - 顺序功能图数据类型 (SFC)。
- 一个可选的描述性注释 (最多 1024 个字符)。可用的字符对应 ASCII 码的 32 到 255

数据实例的名称

用户通过选择一个符号 (最多 32 个字符) 来引用实例，这个符号必须是唯一的。

一些名称是不能使用的，比如说：
- 在文本语言中所使用的关键字，
- 程序代码段的名称，
- 由用户事先定义或选择的数据类型的名称 (结构，数据表)，
- 由用户事先定义或选择的 DFB/EFB 数据类型的名称，
- 由用户事先定义或选择的基本功能 (EF) 的名称。

属于 SFC 系列的实例的名称

在用户设计顺序功能图的时候，实例的名称会隐式地被声明。它们是由厂商提供的缺省名称，用户可以修改这个名称。
厂商提供的缺省名称：

SFC 对象	名称
步	S_< 代码段名称 >_< 步号 >
宏步的步	S_< 代码段名称 >_< 宏步号 >_< 步号 >
宏步	MS_< 代码段名称 >_< 步号 >
嵌套的宏步	MS_< 代码段名称 >_< 宏步号 >_< 步号 >
宏步的输入步	S_IN< 代码段名称 >_< 宏步号 >
宏步的输出步	S_OUT< 代码段名称 >_< 宏步号 >
转换	T_< 代码段名称 >_< 转换号 >
宏步的转换	T_< 代码段名称 >_< 宏步号 >_< 转换号 >

属于功能块系列的实例的名称

在用户把实例插入到应用程序代码段中去的时候，实例的名称会隐式地被声明。它们是由厂商提供的缺省名称，用户可以修改这个名称。
厂商提供的缺省名称的语法：

```
FB_<name of function block type>_<instance No.>
```

注意：实例名称不包括使用实例的代码段的名称，因为它可以用于应用程序的不同代码段中。

访问 DDT 系列实例的元素

访问的语法如下所示：

```
For structure type data
<Instance name>.<Element name>

For table type data
<Instance name>[Element index]
```

规则：
访问语句最多可以有 1024 个字符，导出数据类型可能的限制范围如下所示：
- 10 个嵌套级别 (数据表 / 结构)，
- 每个数据表有 6 维，
- 定义数据表元素索引的 4 位 (数字)。

直接寻址数据实例

内容预览

什么是直接寻址数据实例？(参见*直接寻址数据实例，244 页*)

访问语法

直接寻址数据实例的语法通过 % 符号进行定义，该符号后面有一个内存位置前缀，在某些情况下，还会有一些附加信息。**内存位置前缀**可以是：

- **M**，用于内部变量，
- **K**，用于常数 (Premium)，
- **S**，用于系统变量，
- **N**，用于网络变量，
- **I**， 用于输入变量，
- **Q**，用于输出变量。

%M 内部变量

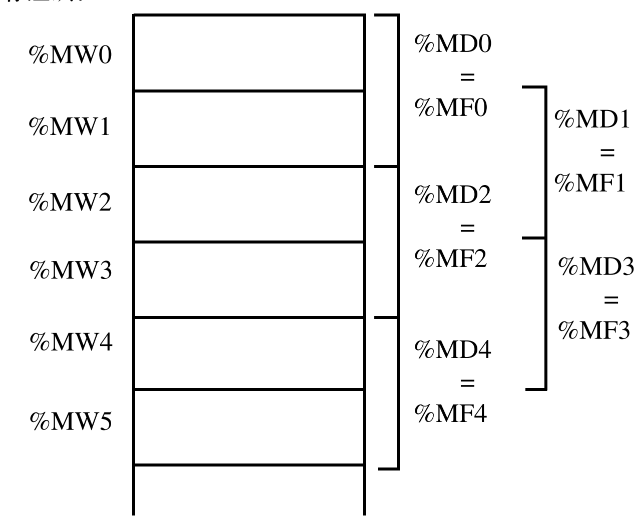
访问语法：

	语法	格式	例子	程序访问权限
位	%M<i> 或 %MX<i>	3 位 (Ebool)	%M1	读 / 写
字	%MW<i>	16 位 (Int)	%MW10	读 / 写
字提取的位	%MW<i>.<j>	1 位 (Bool)	%MW15.5	读 / 写
双字	%MD<i>	32 位 (Dint)	%MD8	读 / 写
实数 (浮点)	%MF<i>	32 位 (Real)	%MF15	读 / 写

<i> 代表实例号 (对 Premium 从 0 开始，对 Quantum 从 1 开始)。

注意：%M<i> 或者 %MX<i> 检测边界值并可进行强制管理。

内存组织：





警告

如果修改 %MW<i>，那么相应的 %MD<i> 和 %MF<i> 也要修改。

如果不遵守此处的警告，可能会导致死亡，严重的伤害，或者设备损伤。

%K 常数

访问语法：

	语法	格式	程序访问权限
字常数	%KW<i>	16 位 (Int)	读
双字常数	%KD<i>	32 位 (Dint)	读
实数 (浮点) 常数	%KF<i>	32 位 (Real)	读

<i> 代表实例号。

注意：内存组织与内部变量的情况是一样的。要注意这些变量在 Quantum PLC 上不可用。

%I 常数

访问语法：

	语法	格式	程序访问权限
位常数	%I<i>	3 位 (Ebool)	读
字常数	%IW<i>	16 位 (Int)	读

<i> 代表实例号。

注意：这些数据仅在 Quantum PLC 上可用。

%S 系统变量

访问语法：

	语法	格式	程序访问权限
位	%S<i> or %SX<i>	1 位 (Bool)	读 / 写或读
字	%SW<i>	32 位 (Int)	读 / 写或读
双字	%SD<i>	32 位 (Dint)	读 / 写或读

<i> 代表实例号。

注意：内存组织与内部变量的情况是一样的。 %S<i> 和 %SX<i> 数据不用来检测边界值，也不能进行强制管理。

%N 网络变量

这些变量包含要在若干个应用程序之间通过通信网络进行交换的信息。

访问语法：

	语法	格式	程序访问权限
公共字	%NW<n>.<s>.<d>	16 位 (Int)	读 / 写或读
字提取的位	%NW<n>.<s>.<d>.<j>	1 位 (Bool)	读 / 写或读

<n> 表示网络号。
<s> 表示站号。
<d> 表示数据号。
<j> 表示位在字中的位置。

输入 / 输出变量的情况

这些变量包含在应用程序专用模块中。

访问语法：

	语法	例子	程序访问权限
输入 / 输出结构 (IODDT)	%CH<@mod>.<c>	%CH4.3.2	读
%I 输入			
Bool 类型模块错误位	%I<@mod>.MOD.ERR	%I4.2.MOD.ERR	读
Bool 类型通道错误位	%I<@mod>.<c>.ERR	%I4.2.3.ERR	读
Bool 或 Ebool 类型位	%I<@mod>.<c>	%I4.2.3	读
	%I<@mod>.<c>.<d>	%I4.2.3.1	读
Int 类型字	%IW<@mod>.<c>	%IW4.2.3	读
	%IW<@mod>.<c>.<d>	%IW4.2.3.1	1¡
Dint 类型双字	%ID<@mod>.<c>	%ID4.2.3	读
	%ID<@mod>.<c>.<d>	%ID4.2.3.1	读
Real 类型实数 (浮点)	%IF<@mod>.<c>	%IF4.2.3	读
	%IF<@mod>.<c>.<d>	%IF4.2.3.1	读
%Q 输出			
Ebool 类型位	%Q<@mod>.<c>	%Q4.20.3	读 / 写
	%Q<@mod>.<c>.<d>	%Q4.20.30.1	读 / 写
Int 类型字	%QW<@mod>.<c>	%QW4.2.3	读 / 写
	%QW<@mod>.<c>.<d>	%QW4.2.3.1	读 / 写
Dint 类型双字	%QD<@mod>.<c>	%QD4.2.3	读 / 写
	%QD<@mod>.<c>.<d>	%QD4.2.3.1	读 / 写
Real 类型实数 (浮点)	%QF<@mod>.<c>	%QF4.2.3	读 / 写
	%QF<@mod>.<c>.<d>	%QF4.2.3.1	读 / 写
%M 变量 (Premium)			
Int 类型字	%MW<@mod>.<c>	%MW4.2.3	读 / 写
	%MW<@mod>.<c>.<d>	%MW4.2.3.1	读 / 写
Dint 类型双字	%MD<@mod>.<c>	%MD4.2.3	读 / 写
	%MD<@mod>.<c>.<d>	%MD4.2.3.1	读 / 写
Real 类型实数 (浮点)	%MF<@mod>.<c>	%MF4.2.3	读 / 写
	%MF<@mod>.<c>.<d>	%MF4.2.3.1	读 / 写

	语法	例子	程序访问权限
%K 常量 (Premium)			
Int 类型字	%KW<@mod>.<c>	%KW4.2.3	R
	%KW<@mod>.<c>.<d>	%KW4.2.3.1	R
Dint 类型双字	%KD<@mod>.<c>	%KD4.2.3	R
	%KD<@mod>.<c>.<d>	%KD4.2.3.1	R
Real 类型实数 (浮点)	%KF<@mod>.<c>	%KF4.2.3	R
	%KF<@mod>.<c>.<d>	%KF4.2.3.1	R

<@mod = \<b>.<e>\<r>.<m>

<b> 总线号 (如果站是本地的，则被忽略)。

<e> 设备连接点号 (如果站是本地的，则被忽略，连接点也称为 Quantum 用户的分站)。

<r> 机架号。

<m> 模块插槽。

<c> 通道号 (0 到 999) 或者 MOD 保留的字。

<d> 数据号 (0 到 999) 或者 ERR 保留的字。

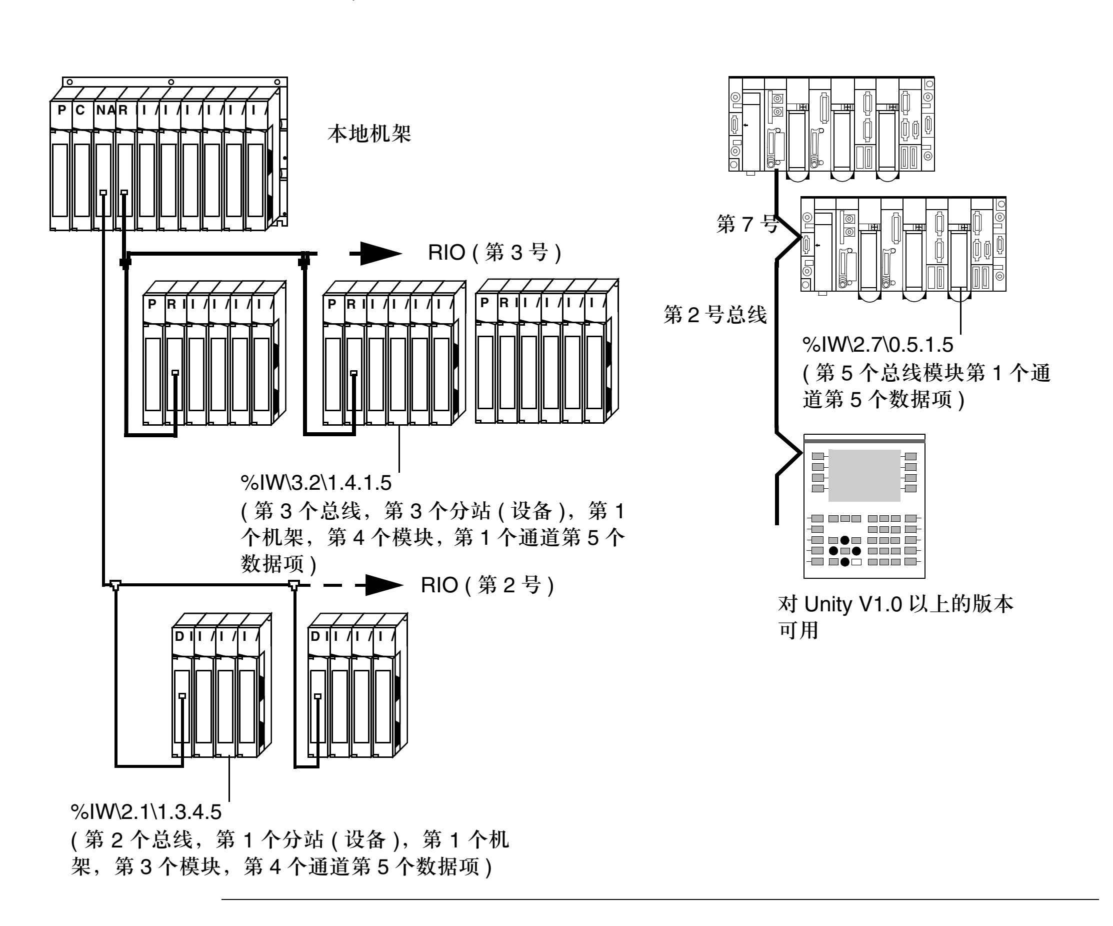
例子：用于 Quantum 和 Premium PLC 的本地站和总线上的站。

Quantum 例子

Premium 例子

(第 0 个机架，第 4 个模块第 1 个通道第 5 个数据项)

%IW1.4.1.5 (第 1 个机架，第 4 个模块第 1 个通道第 5 个数据项)



数据引用

10

内容预览

本章内容

本章描述了数据实例的引用。这些引用包括：
- 基于数值的引用，
- 基于名称的引用，
- 基于地址的引用。

本章内容

本章包含以下内容：

通过数值引用数据实例

介绍

什么是数据实例的引用？(参见*数据引用概述，186 页*)

内容预览

通过数值引用的数据实例，是一个不含有名称 (符号) 或拓扑地址的实例。

它相当于分配给属于 EDT 系列的数据类型实例一个**立即值**。

标准的 IEC 1131 允许在以下数据类型的实例中使用立即值：
- 布尔类型，比如说：
  - BOOL，
  - EBOOL。
- 整数类型，
  - INT，
  - UINT，
  - DINT，
  - UDINT，
  - TIME。
- 实数类型
  - REAL。
- 日期和时间，
  - DATE，
  - DATE AND TIME，
  - TIME OF DAY。
- 字符串。
  - STRING。

编程软件在这些标准之外，还可以添加**位串类型**。
  - BYTE，
  - WORD，
  - DWORD。

立即值的例子：

下面的表格显示了立即值和实例类型之间的关联。

立即值	实例类型
'I am a Character string'	字符串
T#1s	时间
D#2000-01-01	日期
TOD#12:25:23	时钟
DT#2000-01-01-12:25:23	日期和时钟
16#FFF0	字
UINT#16#9AF (类数值)	无符号整数
DWORD#16#FFFF (类数值)	双字

通过名称引用数据实例

介绍

什么是数据实例的引用？(参见*数据引用概述，186 页*)

引用 EDT 系列的实例

用户选择一个名称 (符号) 来访问数据实例：

```
Valve_State: BOOL

Upper_Threshold: EBOOL AT %M10

Hopper_Content: UINT

Oven_Temperature: INT AT %MW100

Encoder_Value: WORD
```

引用 DDT 系列的实例

数据表：
用户选择一个名称 (符号) 来访问数据实例：

```
Giving 2 types of table:

Color_Range ARRAY[1..15]OF STRING
Vehicles ARRAY[1..100]OF Color_range
;

Car: Vehicles
Instance name of the Vehicles-type table

Car[11, 5]
Access to car 11 of the color corresponding to element 5 of
the Color_range table
```

结构：

用户选择一个名称 (符号) 来访问数据实例：

```
Giving the 2 structures:

ADDRESS
   Street: STRING[20]
   Post_Code: UDINT
   Town: STRING: [20]

IDENT
   Surname: STRING[15]
   First name: STRING[15]
   Age: UINT
   Date_of_Birth: DATE
   Location: ADDRESS


Person_1 :IDENT
;Instance name of structure of IDENT type

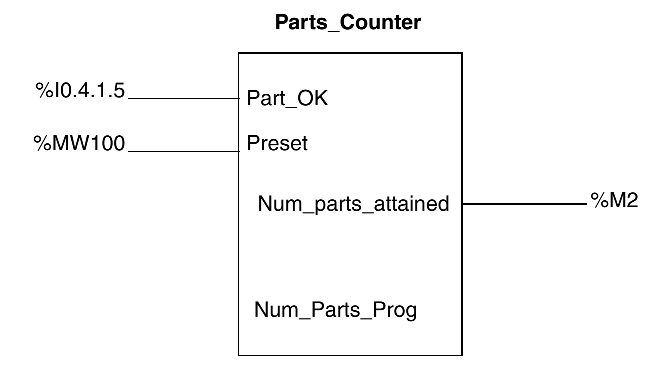
Person_1.Age
Access to the age of Person_1

Person_1.Location.Town
;Access to the location where Person_1 resides
```

引用 DFB\EFB 系列的实例

用户选择一个名称 (符号) 来访问数据实例。

Giving the DFB type:



Screw_Counter: Parts_Counter
Instance name of block of Parts_Counter type

Screw_Counter.Num_Parts_Prog
Access to public variable Num_Parts_Prog

Screw_Counter.Num_parts_attained
Access to the output interface Num_parts_attained

通过地址引用数据实例

介绍

什么是数据实例的引用？(参见*数据引用概述，186 页*)

内容预览

只有属于 **EDT 系列**的特定数据实例，才能通过地址进行引用。这些实例包括：
- 内部变量 (%M<i>， %MW<i>， %MD<i>， %MF<i>)，
- 常数 (%KW<i>， %KD<i>， %KF<i>，
- 输入 / 输出 (%I<address>， %Q<address>)。

通过直接寻址进行引用

如果实例的地址是固定的，那么我们就认为寻址是直接的。

例子：

```
%M1
Access to first bit of the memory

%MW12
Access to twelfth word of the memory

%MD4
Access to fourth double word of the memory

%KF100
;Access to hundredth floating pointing word of the memory

%Q0.4.0.5
Access to fifth bit of the output module in position 4
 of rack 0
```

通过索引地址进行引用

如果实例的地址通过一个索引来完成，我们就认为寻址方式是间接的。

索引可以通过以下内容之一进行定义：
- 一个属于整数类型的数值，
- 一个由整数类型构成的算术表达式。

一个索引变量总是拥有一个非索引的等价变量：

```
%MW<i>[<index>] <=> %MW<j>
```

计算 <j> 的规则如下

对象 <i>[索引]	对象 <j>
%M<i>[对象]	<j>=<i> + < 对象 >
%MW<i>[对象]	<j>=<i> + < 对象 >
%KW<i>[对象]	<j>=<i> + < 对象 >
%MD<i>[对象]	<j>=<i> + (< 对象 > x 2)
%KD<i>[对象]	<j>=<i> + (< 对象 > x 2)
%MF<i>[对象]	<j>=<i> + (< 对象 > x 2)
%KF<i>[对象]	<j>=<i> + (< 对象 > x 2)

例子：

```
%MD6[10] <=> %MD26

%MW10[My_Var+8] <=> %MW20 (with My_Var=2)
```

在编程的过程中，要检查确认如下内容
- 索引不是负值，
- 索引没有超过分配给这三种数据类型的内存空间。

字提取位

可以提取单字 (%MW， %SW ； %KW， %IW， %QW) 的 16 位中的一位。
实例的地址通过提取位 (<j>) 的次序来完成：

```
WORD<i> . <j>
```

例子：

%MW10.4
```
Bit No. 4 of word %MW10
```

%SW8.4
```
Bit No. 4 of system word %SW8
```

%KW100.14
```
Bit No. 14 of constant KW100
```

%QW0.5.1.0.10
```
Bit No. 10 of word 0 of channel 1 of output module 5 of rack 0
```

位和字数据表

有一系列同样类型的，而且具有同样长度的相邻对象 (位或字)

OBJECT<i> :L

位数据表介绍：

类型	地址	写地址
离散量 I/O 输入位	%Ix.i:L	否
离散量 I/O 输出位	%Qx.i:L	是
内部位	%Mi:L	是

字数据表介绍：

类型	地址	写地址
内部字	%MWi:L %MDi:L %MFi:L	是
常数字	%KWi:L %KDi:L %KFi:L	否
系统字	%SW50:4	是

例子：

%M2:65

Defines an EBOOL table from %M2 to %M66

%MW125:30

Defines an INT table from %MW125 to %MW 154

数据命名规则

介绍

在应用程序中，用户要选择一个名称，以便于：

- 定义一个数据类型，
- 对一个数据项 (符号) 进行实例化，
- 一个代码段的识别符。

名称要使用已经定义的规则来避免发生冲突。这意味着有必要把应用程序数据分成不同的区域。

什么是域 ?

它是应用程序的一个区域，用户可以从该区访问一个变量，或者无法访问一个变量，比如说：

- 应用程序的域，包含以下内容：
  - 各种应用程序任务，
  - 组成它的代码段。
- 每一种数据类型的域，比如：
  - 用于 DDT 系列的结构 / 数据表，
  - 用于功能块系列的 EFB/DFB

规则

下面的数据表定义了是否能够把一个已经存在于应用程序的**名称**用于新创建的元素中：

应用程序 --> 新元素 (下面)	代码段	数据表 / 结构类型	EFB/DFB 类型	EF	实例
代码段	否	否	是	是	否
数据表 / 结构类型	否	否	是	是	是
EFB/DFB 类型	是	是	否	否	否 **(3)**
EF	是	是	否	否	否
实例	否	是	是	否 **(1)** 是 **(2)**	否

(1)：属于应用程序域的实例不能和 EF 具有同样的名称。

(2)：属于类型域 (内部变量) 的实例可以和 EF 具有同样的名称。相关的 EF 不能用于这种类型。

(3)：禁止创建或导入与已有实例同名的 EFB/DFB。

注意：除了在上表中给出的规则以外，还有一些如下所述的附加注意事项：
- 在一个类型中，实例 (内部变量) 不能和它所属的对象类型同名，
- 在属于应用程序代码段的实例名称和属于 DFB 代码段的实例名称之间没有冲突，
- 在属于任务的代码段名称和属于 DFB 的代码段名称之间没有冲突。

编程语言



介绍

本节内容

本部分描述了可用编程语言的语法。

本部分内容

本部分包含以下内容：

功能块语言 FBD

11

介绍

概述

本章描述了遵循 IEC-61131 的功能块语言 FBD。

本章内容

本章包含以下内容：

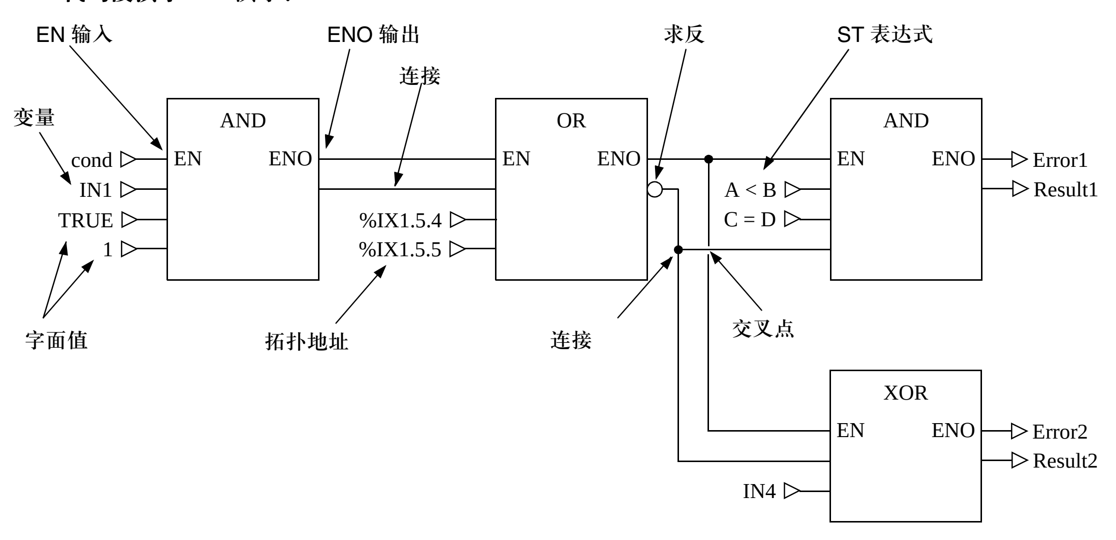
关于 FBD 功能块的常规信息

介绍

FBD 编辑器用于根据 IEC 61131-3 标准进行图形化功能块的编程。

FBD 代码段演示

演示：



对象

编程语言 FBD (功能块图) 的对象把一个代码段划分为若干个：

- EF 和 EFB (基本功能 (参见*基本功能，270 页*) 和基本功能块 (参见*基本功能块，271 页*))，
- DFB (导出功能块) (参见 *DFB，272 页*)，
- 功能程序 (参见*功能程序，272 页*) 以及
- 控制元素 (参见*控制元素，279 页*)。

这些对象通过 FFB 名称进行组合，它们可以通过以下方式彼此建立链接：

- 链接 (参见*链接，280 页*) 或者
- 实际参数 (参见*参数，273 页*)。

可以通过文本对象 (相关的主题*文本对象，282 页*) 为代码段的逻辑提供注释。

代码段大小

一个 FBD 代码段含有一个单页面窗口。
该页码带有灰色的背景。一个网格单元含有 10 个坐标点。在一个 FBD 代码段中，网格单元是两个对象之间最小的间距。
FBD 编程语言不是面向单元的，不过对象仍然可以通过网格来对齐。
一个 FBD 代码段有 300 个水平网格坐标点 (=30 个网格单元) 和 230 个垂直网格坐标点 (=23 个网格单元)。

遵循的 IEC 标准

关于 FBD 编程语言所遵循的 IEC 标准，请参见 *"遵循的 IEC 标准"*。

基本功能，基本功能块，导出功能块和功能程序 (FFB)

介绍

FFB 是以下各项的统称：

- 基本功能 (EF) (参见*基本功能，270 页*)
- 基本功能块 (EFB) (参见*基本功能块，271 页*)
- DFB (导出功能块) (参见 *DFB，272 页*)
- 功能程序 (参见*功能程序，272 页*)

基本功能

基本功能 (EF) 没有内部状态。如果输入值是相同的，那么每次执行该功能的输出就都是一样的 (比如针对两个数值相加的操作，所有功能执行都会给出相同的结果)。

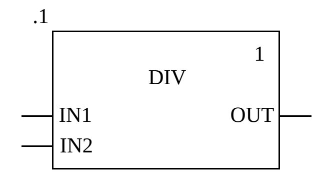
基本功能可以以图形的方式表示为一个框架，它带有输入和一个输出。输入总是位于框架的左侧，输出则总是位于框架的右侧。

功能的名称，亦即功能类型，会在框架的中间给出。

功能的执行号 (参见 *FFB 执行顺序，283 页*) 在功能类型的右侧给出。

功能计数器会在框架的上方给出。功能计数器是当前代码段中的功能序号。功能计数器不能被修改。

基本功能



对于某些基本功能来说，可以增加输入的数量。

基本功能块

基本功能块 (EFB) 具有内部状态。如果输入值相同，每个功能执行的输出可能都是不一样的 (比如对一个计数器来说，输出的数值是递增的)。

基本功能块以图形的方式表示为一个框架，它带有输入和输出。输入总是位于框架的左侧，输出则总是位于框架的右侧。

功能块可以有多个输出。

功能块的名称，亦即功能块类型，会在框架的中间给出。

功能块的执行号 (参见 *FFB 执行顺序，283 页*) 在功能块类型的右侧给出。

实例的名称在框架的上方给出。

实例名称在项目中充当功能块的唯一标识。

实例名称是自动创建的，它具有以下结构：`FBI_n`

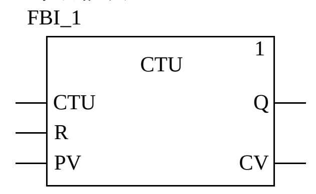
`FBI` = 功能块实例

`n` = 项目中的功能块的序号

这个自动生成的名称可以被修改，以便于用户识别。实例名称 (最多 32 个字符) 在项目中必须是唯一的，不区分大小写。实例名称必须遵循常规命名惯例。

> **注意：**根据 IEC61131-3，实例名称的第一个字符只能是字母。如果您想要使用一个数字作为第一个字符，必须激活该功能。

基本功能块

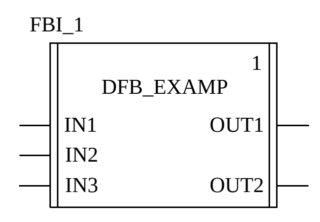


DFB

导出功能块 (DFB) 和基本功能块具有同样的特性。用户可以用编程语言 FBD， LD，IL 和 / 或 ST 来创建它们。

它和基本功能块的唯一区别就在于导出功能块用一个带有垂直双线的框架来表示。

导出功能块



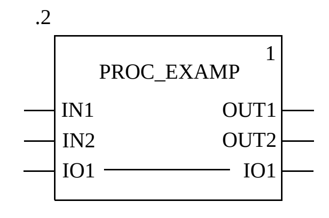
功能程序

功能程序是着眼于技术层面的功能。

它和基本功能的唯一区别就在于功能程序可以占据多个输出，并且支持数据类型 VAR_IN_OUT。

功能程序是 IEC 61131-3 的补充内容，必须被明确激活。除此之外，功能程序和基本功能没有什么区别。

功能程序

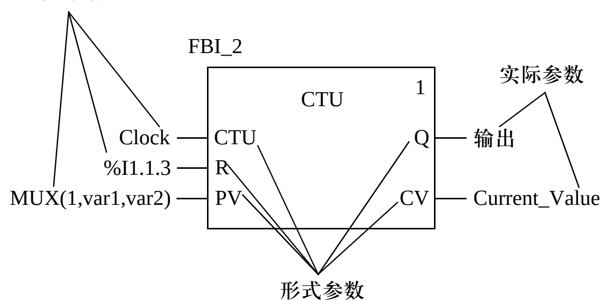


参数

在向 FFB 传递数值和从 FFB 接收数值的过程中，需要用到输入和输出。它们被称为形式参数。

对象被连到包含当前进程状态的形式参数上。它们也称为实际参数。

形式参数和实际参数：



在程序的运行时，来自进程的数值通过实际参数传递到 FFB，在经过处理以后又重新传回输出。

只有下面的类型，可以作为一个对象 (实际参数) 链接到 FFB 输入：

- 变量
- 地址
- 字面值
- ST 表达式 (参见*表达式，425 页*)

连接到 FFB 输入的 ST 表达式是 IEC 61131-3 的附加内容，必须被明确激活。

- 链接

下面对象 (实际参数) 的组合可以和 FFB 输出进行链接：

- 一个变量
- 一个变量和一个或多个连接
- 一个地址
- 一个地址和一个或多个连接
- 一个或多个连接

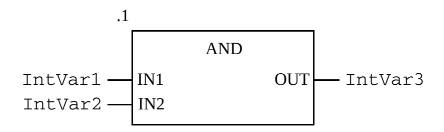
要连接的对象的数据类型必须与 FFB 输入 / 输出的数据类型相匹配。如果所有实际参数都由字面值组成，那么就要为功能块选择一个合适的数据类型。

例外：对于带有 `ANY_BIT` 数据类型的泛型 FFB 输入 / 输出，可以把对象与数据类型 `INT` 或者 `DINT` (不是 `UINT` 和 `UDINT`) 建立关联。

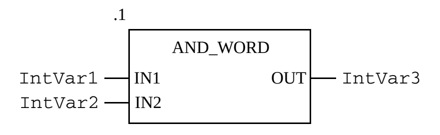
它是 IEC 61131-3 的附加内容，必须被明确激活。

例子：

允许的：



不允许的：



(在这种情况下，必须使用 AND_INT 。)

并不是所有形式参数都需要分配一个实际参数。求反管脚却不然，它们总要分配一个实际参数。还有一些形式参数类型的情况也是这样。您可以在下表中察看这些类型。

形式参数类型的表格：

参数类型	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
EFB：输入	-	+	+	+	/	+	/	+
EFB: VAR_IN_OUT	+	+	+	+	+	+	/	+
EFB：输出	-	-	+	+	+	-	/	+
DFB：输入	-	+	+	+	/	+	/	+
DFB: VAR_IN_OUT	+	+	+	+	+	+	/	+
DFB：输出	-	-	+	/	/	-	/	+
EF：输入	-	-	+	+	+	+	+	+
EF: VAR_IN_OUT	+	+	+	+	+	+	/	+
EF：输出	-	-	-	-	-	-	/	-
功能程序：输入	-	-	+	+	+	+	+	+
功能程序：VAR_IN_OUT	+	+	+	+	+	+	/	+
功能程序：输出	-	-	-	-	-	-	/	+
+ 需要实际参数								
- 不需要实际参数								
/ 不可用								

在没有接收任何数值分配的输入上使用实际参数的 FFB，会用这些实际参数的初始值进行相关操作。

如果形式参数没有被赋值，那么在执行功能块的时候会使用初始值。如果没有定义初始值，就会使用缺省值 (0)。

如果形式参数没有被赋值，并且功能块 /DFB 被多次实例化，那么后面的实例会使用原来的数值进行操作。

公共变量

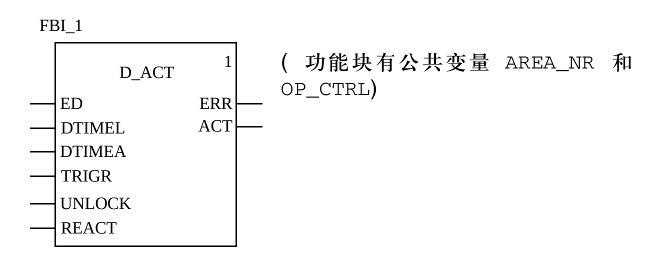
除了输入 / 输出，一些功能块还提供了公共变量。这些变量传递统计数值 (不受进程影响的数值) 传送给功能块。它们用来为功能块设定参数。

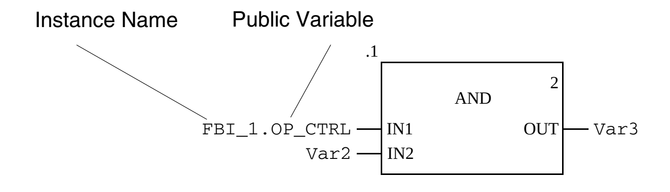
公共变量是 IEC 61131-3 的补充内容。

对公共变量的赋值是通过它们的初始值来完成的。

公共变量可以通过功能块的实例名称和公共变量的名称读出。

例子：





编程注意事项

在编程的过程中应该注意以下事项：

- 只有在输入 EN=1 或者输入 EN 变灰的情况下，才能执行 FFB (请同时参见 *EN 和 ENO，276 页*)。
- 布尔输入和输出可以取反。
- 在使用 VAR_IN_OUT 变量的时候，可以应用特殊的条件 (参见 VAR_IN_OUT *变量，277 页*)。
- 功能块 /DFB 实例可以多次调用 (请同时参见 *多重功能块实例调用，276 页*)。

多重功能块实例调用

功能块 /DFB 实例可以多次调用；相比之下，来自通信 EFB 和功能块 /DFB 的带有一个 ANY 输出，而不带有 ANY 输入的实例，则只能调用一次。

也可以多次调用同一个功能块 /DFB 实例，比如下面的情况：

- 如果功能块 /DFB 没有内部数值，或者在后面的处理中不需要用到它。
  在这种情况下，因为功能块 /DFB 的代码只加载一次，所以通过多次调用同一个功能块 /DFB 实例，内存会被保存起来。接下来功能块 /DFB 会像一个 “功能” 那样被处理。
- 如果功能块 /DFB 带有一个内部数值，并且该数值会影响到各个程序段，比如说，计数器的数值应该在程序的各个部分增加。
  在这种情况下，调用同一个功能块 /DFB 时，不必为将来在程序的其他部分所进行的处理保存当前结果。

EN 和 ENO

在所有 FFB 中都可以使用一个 EN 输入和一个 ENO 输出。

如果在调用 FFB 的时候， EN 的数值等于 “0”，由 FFB 所定义的算法就不会执行， ENO 会置为 “0”。

如果在调用 FFB 的时候， EN 的数值等于 “1”，由 FFB 所定义的算法会执行。在成功执行这些算法以后， ENO 会置为 “1”。如果在执行这些算法的时候发生了错误， ENO 会置为 “0”。

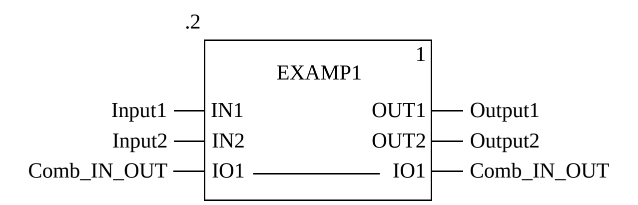
如果 ENO 置为 “0” (因为 EN=0 或者执行过程中发生错误)，

- 功能块输出保持上一个周期它们正确执行时的状态。
- 功能和功能程序输出置为 “0”。

FFB 的输出结果，在无 EN/ENO 和 EN=1 时，是一样的。

VAR_IN_OUT 变量

FFB 经常用来把一个输入处的读入值 (输入变量)，处理该变量，并把同一个变量名的变化值 (输出变量) 输出。
这种特殊类型的输入 / 输出变量也称为 VAR_IN_OUT 变量。
在输入和输出变量之间的链接通过 FFB. VAR_IN_OUT 变量中的一条线表示出来。

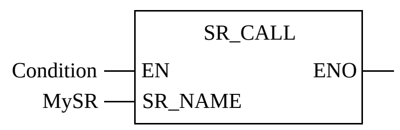


在使用带有 VAR_IN_OUT 变量的 FFB 时，需要注意以下特殊方面：
- 所有 VAR_IN_OUT 输入都必须分配一个变量。
- 不能将图形链接连到 VAR_IN_OUT 输入 / 输出。
- 不能把任何字面值或者常数连到 VAR_IN_OUT 输入 / 输出。
- 不能在 VAR_IN_OUT 输入 / 输出上使用任何取反操作。
- 必须为 VAR_IN_OUT 输入和 VAR_IN_OUT 输出连接同一个变量 / 变量成分。

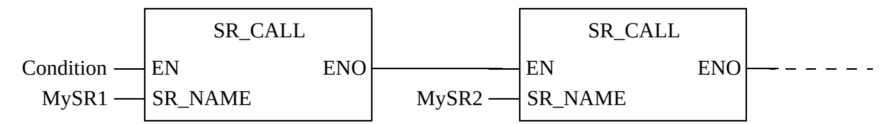
子程序调用

调用子程序

在 FBD 中，子程序通过以下功能块进行调用：



如果 EN 的状态是 1，就会调用相应的子程序 (变量名连接到 SR_Name 的输入端)。
这种功能块不使用输出 ENO 来显示错误状态。它的输出 ENO 总是 1，这个输出用来在同一时间内调用多个子程序。
通过下面的结构，用户可以同时调用多个子程序：



要调用的子程序必须和调用的 FBD 代码段位于同一个任务内。
子程序也可以从子程序内进行调用。
子程序调用 IEC 61131-3 的补充内容，必须被明确激活。
在 SFC 动作代码段中，只有在启动了多令牌操作的情况下，才能进行子程序调用。

控制元素

介绍

控制元素用来在 FBD 代码段中执行跳转，以及从一个子程序 (SRx) 或者导出功能块 (DFB) 返回到主程序。

控制元素

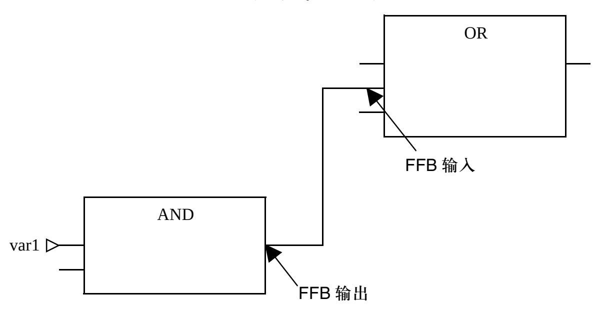
以下为可用的控制元素：

名称	演示	描述
跳转	NEXT ——>>	如果左链接的状态是 1，就会产生一个到标记的跳转 (在当前代码段)。 如果要生成一个有条件跳转，要把一个跳转对象链接到一个布尔 FFB 输出。 如果要生成一个无条件跳转，可以通过诸如 AND 这样的功能把数值 1 赋给跳转对象。
标记	LABEL:	标记 (跳转目标) 以文本表示，其末尾有一个冒号。 这个文本最多可以有 32 个字符，在整个代码段中必须是唯一的。文本必须符合常规命名惯例。 跳转标记只能放在代码段左边缘的头两个网格之间。 **注意：**跳转标记不能 “穿越”网格中的对象，也就是说，任何对象都不能放于从跳转标记到代码段右边缘的母线之间。这对跳转链接也同样适用。
返回	-< RETURN >	每一个子程序和 DFB (导出功能块) 在被处理完毕以后都会退出，也就是说，返回到被调用的主程序。 如果子程序 /DFB 提早离开，可以通过返回对象强制使其返回主程序。 如果左链接的状态是 1，会从子程序或者 DFB (导出功能块) 返回到主程序。 返回对象只能用于 DFB 或者 SR 子程序。它们不能用在主程序内。 如果要生成一个有条件返回，需要把一个返回对象链接到一个布尔 FFB 输出上。

链接

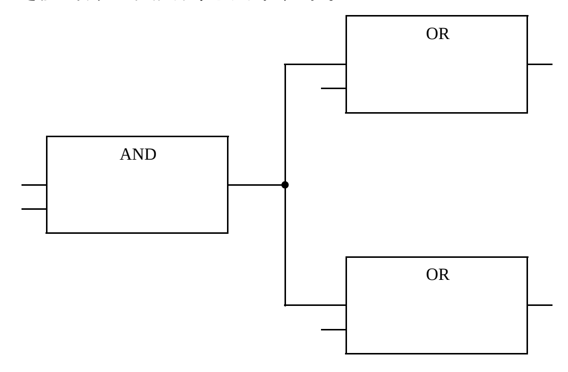
描述

链接是 FFB 之间的垂直和水平连接。

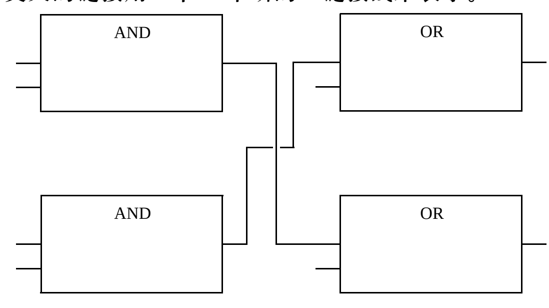


演示

链接的分叉点用实心圆来表示。



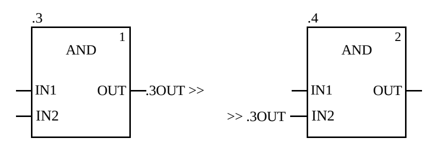
交叉的链接用一个 “中断的”链接线来表示。



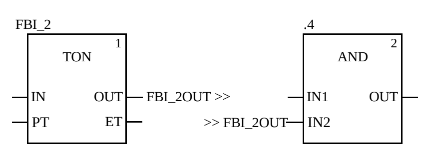
编程注意事项

在编程的过程中，应该注意以下方面：

- 链接可以用于所有数据类型。
- 要链接的输入 / 输出数据类型必须一致。
- 一个 FFB 输出可以做若干个链接。不过一个 FFB 输入只能与一个链接进行连接。
- 输入和输出可以互相链接。不能同时链接多个输出。这表示在 FBD 中不能使用链接来创建 “或” 连接。在这种情况下，要使用一个 OR 功能。
- 可以与其他目标进行重叠链接。
- 不能使用链接来创建环路，因为在这种情况下，在代码段中无法确定执行顺序。要创建环路，必须使用实际参数 (参见*设置循环，290 页*)。
- 为了避免链接之间出现交叉，也可以用连接器来表示链接。
  连接的源和目标都带有一个名称，这些名称在代码段内是唯一的。
  根据用于连接的源对象的类型，连接器名称具有如下结构：
  - 对于功能：在连接源使用 “功能计数器 / 形式参数”



  - 对于功能块：在连接源使用 “实例名称 / 形式参数”



文本对象

描述

在 FBD 功能块语言中可以使用文本对象。这些文本对象的大小取决于文本的长度。对象尺寸由文本大小决定，它可以沿垂直或水平方向扩展，占用更多的网格单元。文本对象不能与 FFB 重叠；不过它们可以与链接重叠。

FFB 的执行顺序

介绍

执行顺序由 FFB 在代码段内的位置决定 (从左到右，从上到下执行)。如果 FFB 以图形的方式链接，执行顺序是由信号流来确定的。

执行顺序由执行号码 (在 FFB 框架右上方的号码) 来表示。

网络执行顺序

对于网络执行顺序，有如下适用的规则：

- 对代码段的执行，是基于每个 FFB 链接，从上至下，逐个网络地完成的。
- 不能使用链接来创建环路，因为在这种情况下，无法准确地确定执行顺序。要创建环路，必须使用实际参数 (参见*设置循环，290 页*)。
- 当图形没有链接来确定的网络执行顺序时 (从右上方到左下方)。这个执行顺序可以改变的 (参见*改变执行顺序，285 页*)。
- 如果后面的网络要用到前面网络的输出，那么在开始处理后面的网络之前，先要结束前面网络的处理。
- 所有网络元素都应该等到它全部的输入状态计算完毕以后，才能进行处理。
- 只有当一个网络的所有输出都被处理完毕，该网络的处理才算结束。

在网络内的信号流

针对一个网络内的执行顺序，有以下适用的规则：

- 只有当与其输入相链接的所有元素 (FFB 输出等) 处理完毕以后，才能处理 FFB。
- 与同一个 FFB 的各种输出相链接的 FFB 的执行顺序应该自上而下。
- FFB 的执行顺序不受它在网络中的位置的影响。

如果多个 FFB 链接到 “调用” FFB 的同一个输出，那么这一条规则就不适用了。在这种情况下，执行顺序由图形顺序决定 (自上而下)。

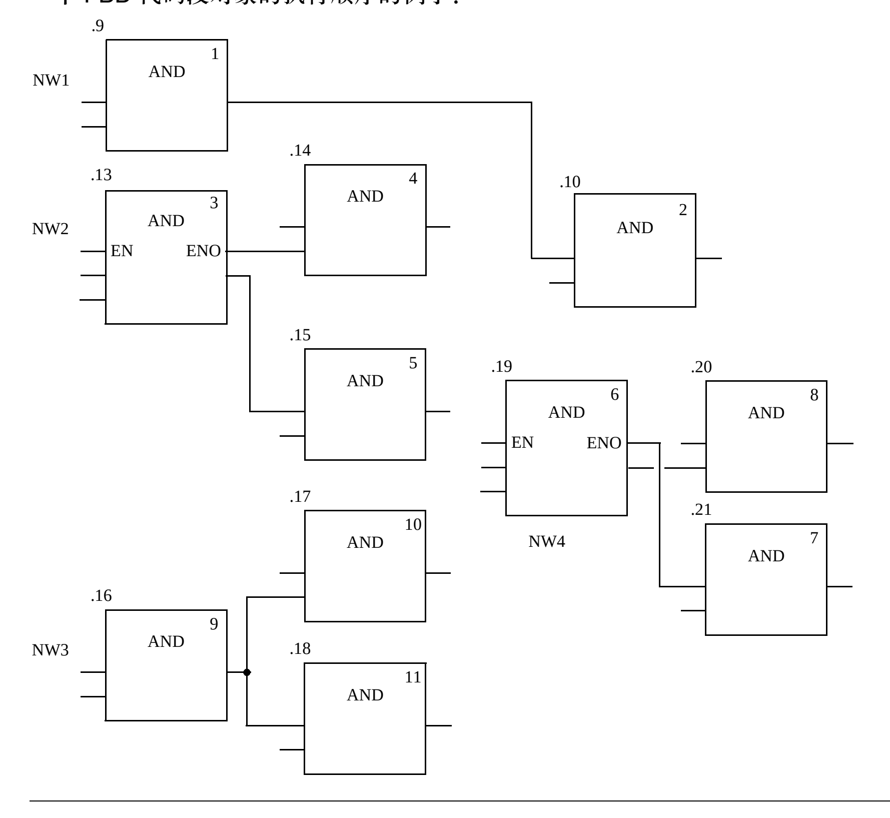
优先级

在定义一个代码段内的信号流时的优先级：

优先级	规则	描述
1	链接	在定义一个 FBD 代码段内的信号流时，链接具有最高的优先级。
2	用户定义	用户确定执行顺序
3	逐个网络进行	在开始处理其他网络之前，要完全结束本网络的处理。
4	输出顺序	链接到同一个 “调用” FFB 的输出的 FFB 会自上而下进行处理。
5	逐个梯级	优先级最低 (只有在其他规则都不适用的时候才使用此规则)。

例子

一个 FBD 代码段对象的执行顺序的例子：



更改执行顺序

介绍

网络的执行顺序和一个网格中的对象的执行顺序，是通过一系列规则进行定义的 (参见 *优先级，284 页*)。

(参见 *优先级，284 页*)

在某些情况下，用户应该更改由系统推荐的执行顺序。定义 / 更改网络执行顺序的功能程序如下所示：

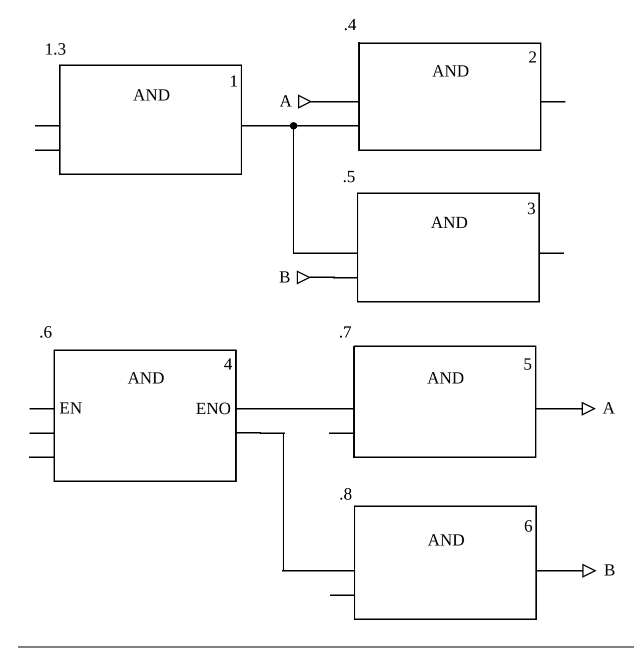
- 使用链接，而不是实际参数
- 网络位置
- 明确的执行顺序定义

定义 / 更改网络执行顺序的功能程序如下所示：

- FFB 位置

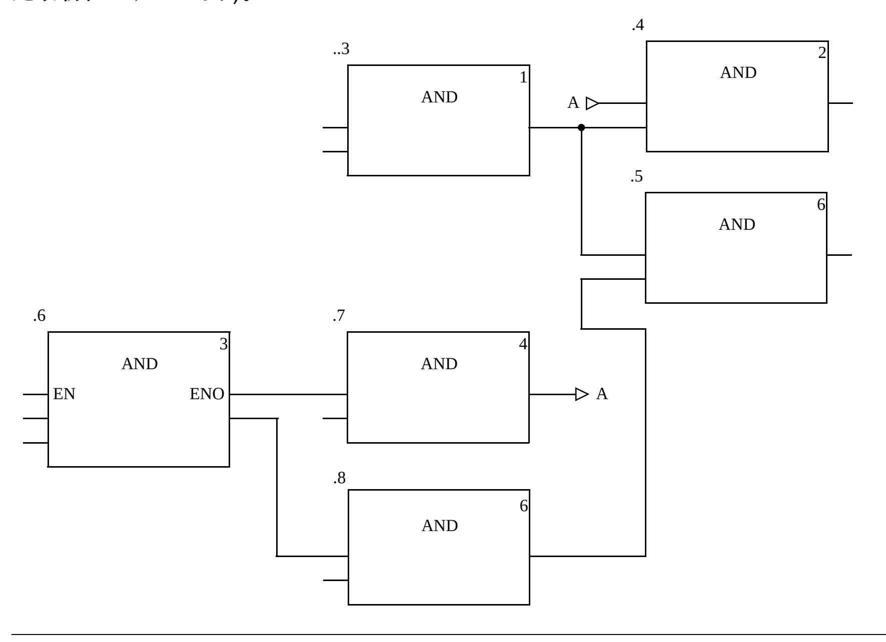
最初位置

在下面的演示中给出了两个网络，它们的执行顺序仅通过它们在代码段中的位置进行定义，并没有考虑到功能块 `0.4/0.5` 和 `0.7/0.8` 需要另外一个执行顺序。



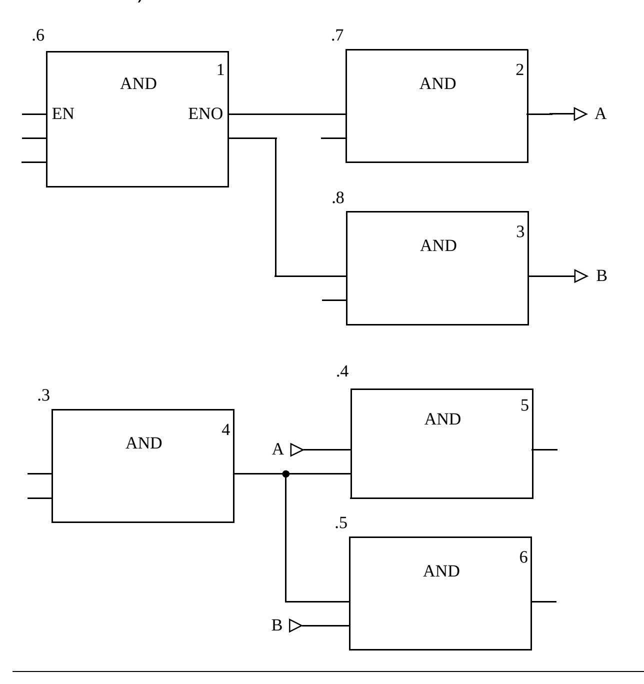
使用链接，而非实际参数

通过使用一个链接，而不是一个变量，两个网络可以以正确的顺序运行 (请同时参见*最初位置，285 页*)。



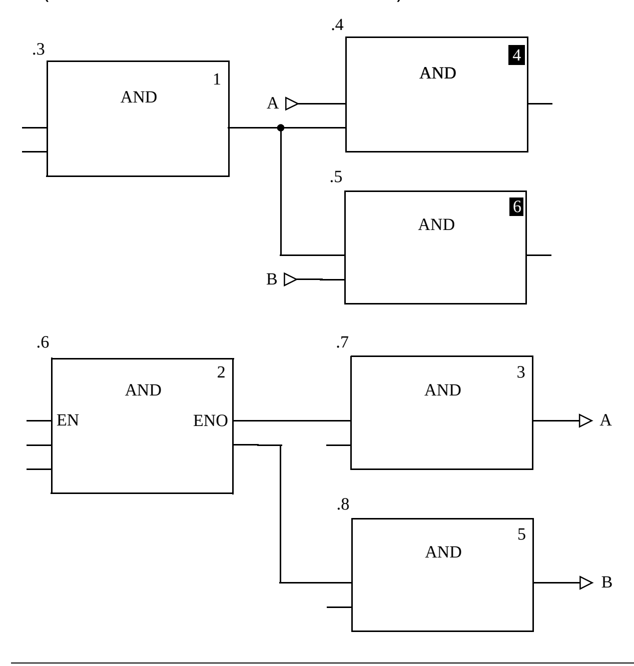
网络位置

可以通过在代码段中更改网络的位置来获得正确的执行顺序 (请同时参见*最初位置，285 页*)。



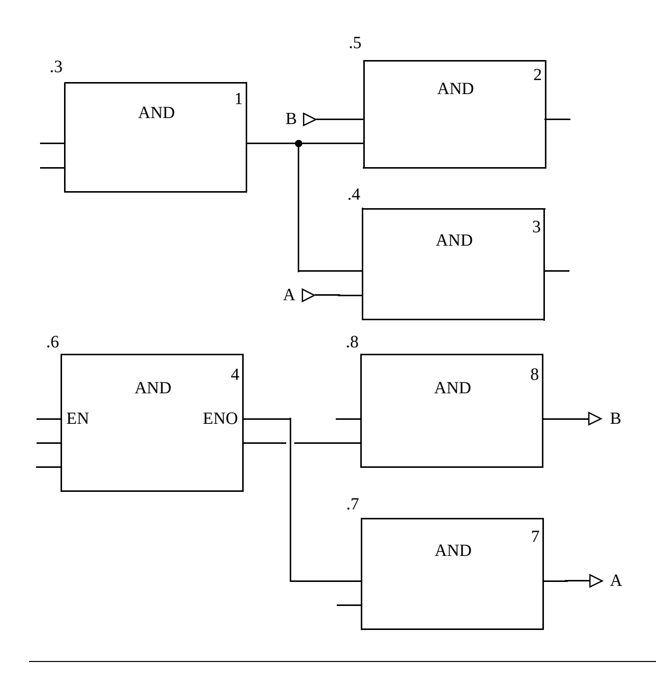
明确的定义

可以通过明确更改 FFB 的执行顺序来获得正确的执行顺序。执行顺序被明确更改的 FFB 会在执行号码前面接收到一个关于先前执行的 FFB 的实例名称 / 功能号码附加项 (请同时参见*最初的位置，285 页*)。



FFB 位置

只有在多个 FFB 同时链接到了 “调用” FFB 的同一个输出时， FFB 的位置才会影响到执行顺序 (请同时参见*最初的位置，285 页*)。在第一个网络中，功能块的位置 `.4` and `.5` 被交换了。在这种情况下 (两个功能块的输入具有公共的来源)，两个功能块的执行顺序也被交换了 (自上而下进行处理)。
在第二个网络中，功能块的位置 `.7` and `.8` 被交换了。在这种情况下 (两个功能块的输入具有不同的来源)，两个功能块的执行顺序没有被交换 (以调入功能块输出的顺序进行处理)。

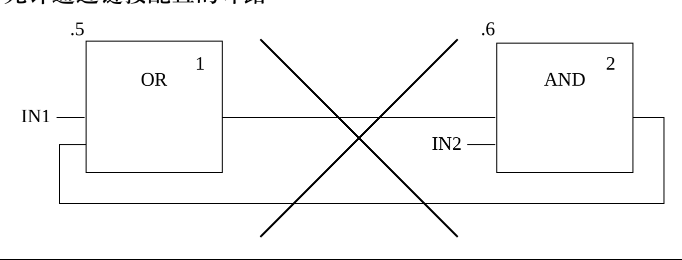


设置循环

不允许的环路

不能只通过链接对环路进行配置，因为无法明确说明信号流 (一个 FFB 的输出是下一个 FFB 的输入，这个 FFB 的输出又是第一个 FFB 的输入)。

不允许通过链接配置的环路



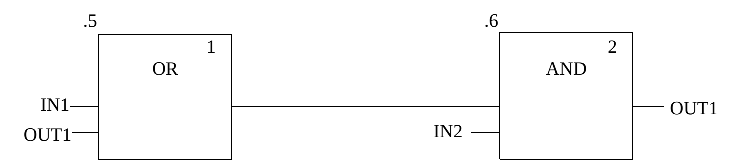
通过一个实际参数生成环路

此类逻辑必须通过反馈变量解决，以便能够确定信号流。

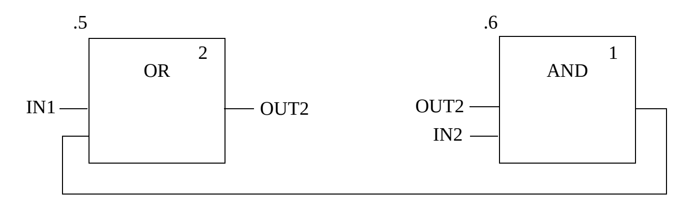
必须对反馈变量进行初始化。初始值用在逻辑的第一次执行中。在它们执行完毕以后，初始值就会被实际值所取代。

请注意用于两种功能块的两种不同的执行顺序 (在实例名称后面的括号中的数值)。

通过一个实际参数生成的环路：第 1 类



通过一个实际参数生成的环路：第 2 类



梯形图 LD

12

介绍

概述

本章描述了遵循 IEC-611311 的梯形图语言 LD。

本章内容

本章包含以下内容：

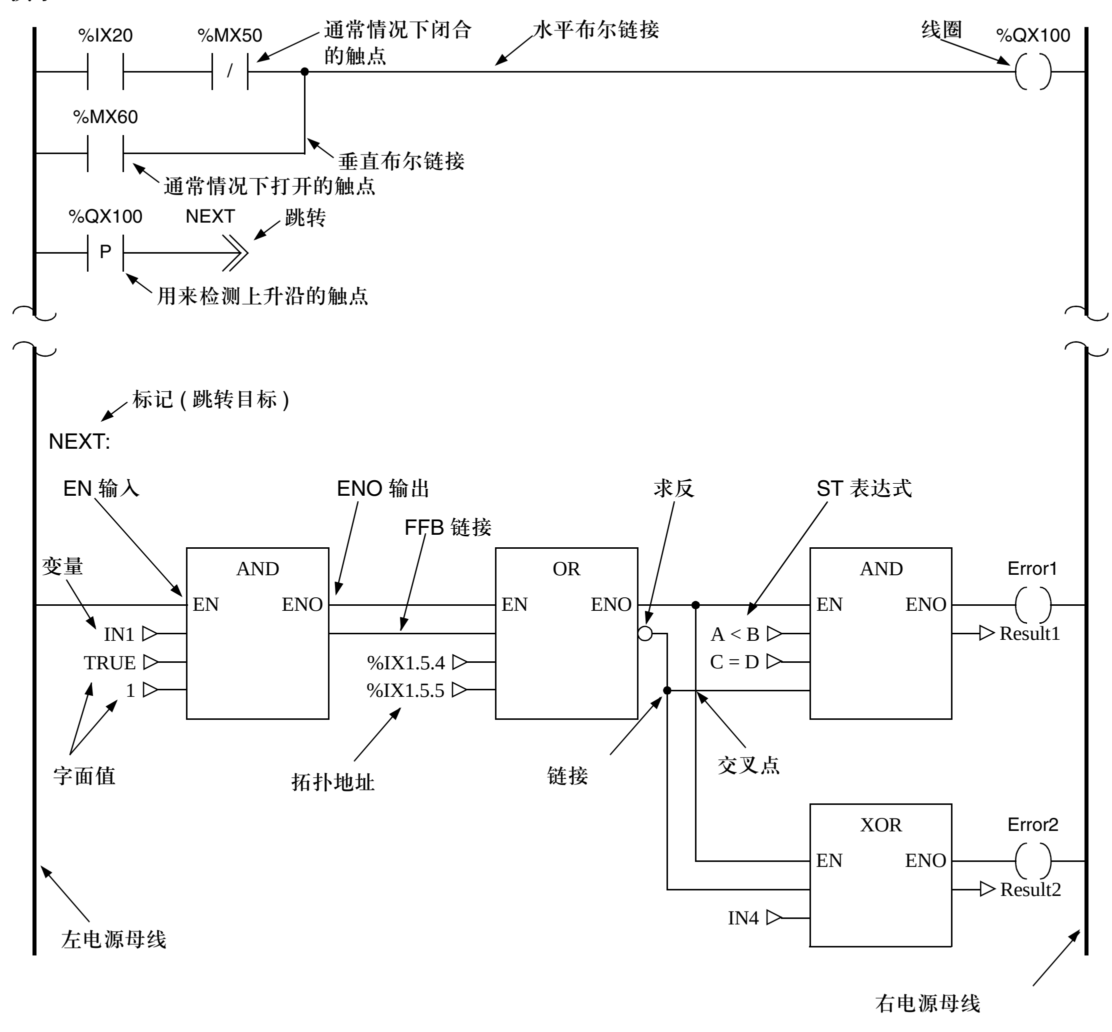
关于 LD 梯形图语言的常规信息

介绍

本节描述了遵循 IEC 61131-3 的梯形图 (LD)。
LD 代码段的结构相当于继电器开关电路的梯级。
左面的电源母线位于 LD 编辑器的左侧。这个左侧的电源母线相当于梯级的相电源 (L 梯形)。通过 LD 编程，与梯级中的情况一样，只有链接到电源，也就是连接到左电源母线的 LD 对象才会被 “处理”。右电源母线相当于中性线。所有线圈和 FFB 输出都和它以直接或者间接的方式链接起来，从而产生一个电流。
一组自上而下链接在一起的对象，被称为一个 网络或者一个梯级。

一个 LD 代码段的演示

演示：



对象

编程语言 LD 的对象把一个代码段划分成若干：

- 触点 (参见*触点，295 页*)，
- 线圈 (参见*线圈，296 页*)，
- EF 和 EFB(基本功能 (参见*基本功能，298 页*) 和基本功能块 (参见*基本功能块，299 页*))，
- DFB(导出功能块 (参见 *DFB，300 页*))
- 功能程序 (参见*功能程序，300 页*)
- 控制元素 (参见*控制元素，306*) 和
- 操作和比较功能块 (参见*操作和比较功能块，307 页*)，它是 IEC 61131-3 的扩展内容

这些对象可以通过以下方式互相连接：

- 链接 (参见*链接，309 页*) 或者
- 实际参数 (参见*参数，301 页*) (仅限于 FFB)。

对于带有文本对象 (参见*文本对象，312 页*) 的代码段逻辑，可以提供注释。

代码段的大小

一个 LD 代码段包含一个带有页码的窗口。

该页有一个网格，这个网格把代码段划分成行和列。

LD 代码段可以定义 11 到 64 列以及 17 到 2000 行。

LD 编程语言是面向单元的，也就是说，在每一个单元中只能放置一个对象。

处理顺序

在 LD 代码段中，个体对象的处理顺序是由代码段中的数据流来决定的。连接到左侧电源母线的网络按照自上而下的顺序进行处理 (与左侧电源母线的链接)。在代码段内互相独立的网络，则按照位置排布的顺序进行处理 (自上而下) (请同时参见*执行顺序和信号流，313 页*)。

遵循的 IEC 标准

关于 LD 编程语言遵循的 IEC 标准，请参见 “遵循的 IEC 标准”。

触点

介绍

触点是一个 LD 元素，它把水平链接上的状态传递到它的右侧。这个状态来自左侧水平链接状态的布尔 AND 链接，具有相关布尔实际参数的状态。

触点不改变相关实际参数的数值。
触点占据一个单元。

下面各项可以作为实际参数使用：

- 布尔变量
- 布尔常数
- 布尔地址 (拓扑地址或者符号地址)
- ST 表达式 (参见*表达式，425 页*)，返回一个布尔结果 (比如 `VarA OR VarB`)

ST 表达式，用于触点的实际参数是 IEC 61131-3 的补充内容，必须被明确激活。

触点类型

以下是可用的触点：

名称	演示	描述
常开	xxx ─┤ ├─	对于通常情况下断开的触点，如果相关布尔实际参数 (用 xxx 表示) 处于接通状态，那么左链接的状态会被传递到右链接，否则右链接会处于断开状态。
常闭	xxx ─┤ / ├─	对于通常情况下闭合的触点，如果相关布尔实际参数 (用 xxx 表示) 处于断开状态，那么左链接的状态会被传递到右链接，否则右链接会处于断开状态。
用来检测上升沿的触点	xxx ─┤ P ├─	对于检测上升沿的触点，如果相关实际参数 (用 xxx 标记) 从断开状态切换到闭合状态，并且与此同时左链接处于接通状态，那么程序周期的右链接就处于闭合状态，否则右链接会处于断开状态。
用来检测下降沿的触点	xxx ─┤ N ├─	对于检测下降沿的触点，如果相关实际参数 (用 xxx 标记) 从接通状态切换到断开状态，并且与此同时左链接处于接通状态，那么程序周期的右链接就处于闭合状态，否则右链接会处于断开状态。

线圈

介绍

线圈是一个 LD 元素，它把左侧水平链接的状态原封不动地传递给右侧的水平链接。状态被存储在相应的布尔实际参数中。

通常情况下，线圈跟随 FFB 的触点，不过它们也可以带动触点。线圈占据一个单元。

下面是可用的实际参数：

- 布尔变量
- 布尔地址 (拓扑地址或者符号地址)

线圈类型

以下是可用的线圈：

名称	演示	描述
线圈	xxx —( )—	通过线圈，左链接的状态被传递到相关的布尔实际参数 (用 xxx 表示) 和右链接中去。
求反线圈	xxx —(/)—	通过求反线圈，左链接的状态被复制到右链接中去。左链接的反转状态被复制到相关的布尔实际参数 (用 xxx 表示) 中去。如果链接处于断开状态，那么右链接也会处于断开状态，相关的布尔实际参数会处于接通状态。
用来检测上升沿的线圈	xxx —(P)—	通过用来检测上升沿的线圈，左链接的状态被复制到右链接中去。如果左链接的转换从断开状态切换到接通状态，那么相关的数据类型 EBOOL 的实际参数 (用 xxx 表示) 对一个程序周期来说处于接通状态。
用来检测下降沿的线圈	xxx —(N)—	通过用来检测下降沿的线圈，左链接的状态被复制到右链接中去。如果左链接的转换从接通状态切换到断开状态，那么相关的数据类型 EBOOL 的实际参数 (用 xxx 表示) 对一个程序周期来说处于接通状态。
置位线圈	xxx —(S)—	通过置位线圈，左链接的状态被复制到右链接中去。如果左链接处于接通状态，那么相关的布尔实际参数 (用 xxx 表示) 会置为接通状态，否则它就保持原状态不变。相关的布尔实际参数只能通过“复位线圈”进行复位。

名称	演示	描述
复位线圈	xxx —(R)—	通过复位线圈，左链接的状态被复制到右链接中去。如果左链接处于接通状态，那么相关的布尔实际参数 (用 xxx 表示) 会复位为断开状态，否则它就保持原状态不变。相关的布尔实际参数只能通过“置位线圈”进行设定。
停止线圈	xxx —(H)—	通过停止线圈，如果左链接处于接通状态，程序就会立即停止执行。(左链接的状态不会通过停止线圈被复制到右链接中去。)
调用线圈	xxx —(C)—	通过调用线圈，左链接的状态被复制到右链接中去。如果左链接处于接通状态，相应的子程序 (用 xxx 表示) 就会被调用。 要调用的子程序必须与调用的 LD 代码段位于同一个任务中。子程序也可以从子程序内部进行调用。 子程序是 IEC 61131-3 的补充内容，必须被明确激活。 在 SFC 动作代码段中，只有在激活了多令牌操作功能的情况下，才能使用调用线圈 (子程序调用)。

基本功能，基本功能块，导出功能块和功能程序 (FFB)

介绍

FFB 是以下各项的统称：
- 基本功能 (EF) (参见*基本功能，298 页*)
- 基本功能块 (EFB) (参见*基本功能块，299 页*)
- 导出功能块 (DFB) (参见 *DFB，300 页*)
- 功能程序 (参见*功能程序，300 页*)

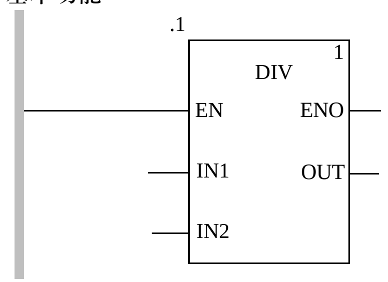
FFB 占据 1 到 3 列 (取决于形式参数名称的长度) 以及 2 到 33 行 (取决于形式参数的行数)。

基本功能

功能没有内部状态。如果输入数值相同，那么这个功能执行的结果都具有相同的数值 (比如针对两个数值相加的操作，这个功能执行都会给出相同的结果)。

基本功能以图形的方式表示为一个框架，它带有输入和一个输出。输入总是位于框架的左侧，输出则总是位于框架的右侧。
功能的名称，亦即功能类型，会在框架的中间给出。
功能的执行号 (参见*执行顺序和信号流， 313 页*) 在功能类型的右侧给出。

功能计数器会在框架的上方给出。功能计数器是当前代码段中的功能序号。功能计数器不能被修改。
基本功能



对于某些基本功能来说，可以增加输入的数量。

基本功能块

基本功能块 (EFB) 具有内部状态。如果输入值相同，每个功能执行的输出可能都是不一样的 (比如对一个计数器来说，输出的数值是递增的)。

基本功能块可以以图形的方式表示为一个框架，它带有输入和输出。输入总是位于框架的左侧，输出则总是位于框架的右侧。功能块的名称，亦即功能块类型，会在框架的中间给出。实例的名称在框架的上方给出。

功能块可以有多个输出。
功能块的名称，也就是功能块类型，会在框架的中间给出。
功能块的执行号 (参见*执行顺序和信号流， 313 页*) 在功能块类型的右侧给出。

实例的名称在框架的上方给出。
实例名称在项目中充当功能块的唯一标识。
实例名称是自动创建的，它具有以下结构：
`FBI_n`

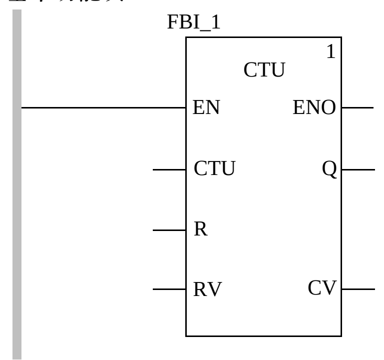
`FBI` = 功能块实例

`n` = 项目中的功能块的序号

这个自动生成的名称可以被修改，以便于用户识别。实例名称 (最多 32 个字符) 在项目中必须是唯一的，不区分大小写。实例名称必须遵循常规命名惯例。

> **注意：**根据 IEC61131-3，实例名称的第一个字符只能是字母。如果您想要使用一个数字作为第一个字符，必须明确地激活该功能。

基本功能块

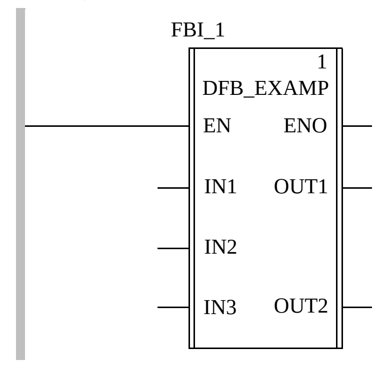


DFB

导出功能块 (DFB) 和基本功能块具有同样的特性。用户可以用编程语言 FBD， LD， IL 和 / 或 ST 来创建它们。

它和基本功能块的唯一区别就在于导出功能块用一个带有垂直双线的框架来表示。

导出功能块



功能程序

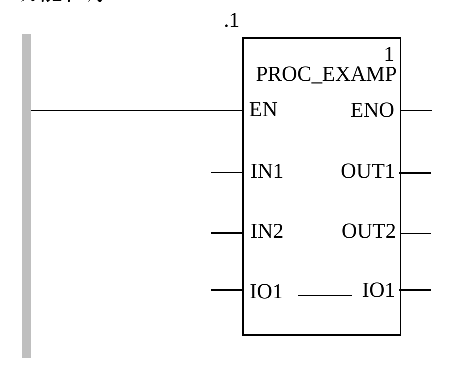
功能程序是着眼于技术层面的功能。

它和基本功能的唯一区别就在于功能程序可以占据多个输出，并且支持数据类型 VAR_IN_OUT。

除此之外，功能程序和基本功能没有什么区别。

功能程序是 IEC 61131-3 的补充内容，必须被明确激活。

功能程序



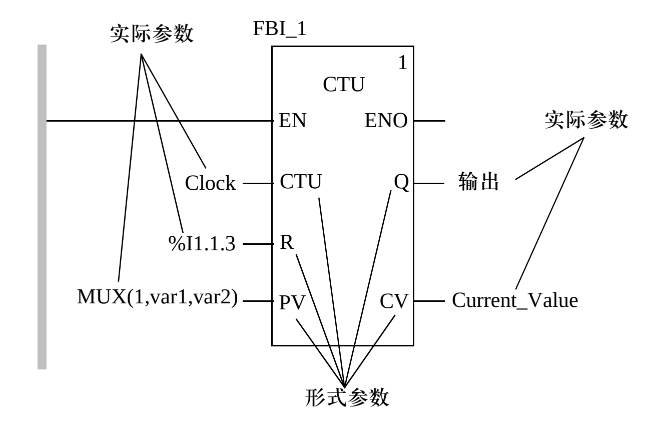
参数

在向 FFB 传递数值和从 FFB 接收数值的过程中，需要用到输入和输出。它们被称为形式参数。

包含当前进程状态的对象被连到形式参数上。

它们被称为实际参数。

形式参数和实际参数：



在程序的运行时内，来自进程的数值通过实际参数传递到 FFB，在经过处理以后又重新传回输出。

下面的一种对象 (实际参数) 可以链接到 FFB 输入：

- 触点
- 变量
- 地址
- 字面值
- ST 表达式

到 FFB 输入的 ST 表达式是 IEC 61131-3 的附加内容，必须被明确激活。

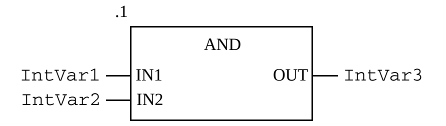
- 链接

下面的对象 (实际参数) 组合可以和 FFB 输出进行链接：

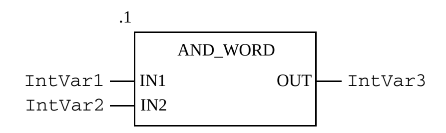
- 一个或多个线圈
- 一个或多个触点
- 一个变量
- 一个变量和一个或多个连接
- 一个地址
- 一个地址和一个或多个连接
- 一个或多个连接

要连接的对象的数据类型必须与 FFB 输入 / 输出的数据类型相匹配。如果所有实际参数都由字面值组成，那么就要为功能块选择一个合适的数据类型。

例外：对于带有 ANY_BIT 数据类型的泛型 FFB 输入 / 输出，可以把对象与数据类型 INT 或者 DINT （不是 UINT 和 UDINT) 建立关联。
它是 IEC 61131-3 的附加内容，必须被明确激活。
例子：
允许的：



不允许的：



(在这种情况下，必须使用 AND_INT)

并不是所有形式参数都需要分配一个实际参数。求反管脚却不然，它们总要分配一个实际参数。还有一些形式参数类型的情况也是这样。您可以在下表中察看这些类型。
形式参数类型的表格：

参数类型	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
EFB：输入	-	+	+	+	/	+	/	+
DFB：输出	-	-	+	/	/	-	/	+
EFB: VAR_IN_OUT	+	+	+	+	+	+	/	+
DFB：输入	-	+	+	+	/	+	/	+
DFB: VAR_IN_OUT	+	+	+	+	+	+	/	+
EFB：输出	-	-	+	+	+	-	/	+
EF：输入	-	-	+	+	+	+	+	+
EF: VAR_IN_OUT	+	+	+	+	+	+	/	+
EF：输出	-	-	-	-	-	-	/	-
功能程序：输入	-	-	+	+	+	+	+	+
功能程序：VAR_IN_OUT	+	+	+	+	+	+	/	+
功能程序：输出	-	-	-	-	-	-	/	+
+ 需要实际参数								
- 不需要实际参数								
/ 不可用								

在没有接收任何数值分配的输入上使用实际参数的 FFB，会用这些实际参数的初始值进行相关操作。

如果形式参数没有被赋值，那么在执行功能块的时候会使用初始值。如果没有定义初始值，就会使用缺省值 (0)。

如果形式参数没有被赋值，并且功能块 /DFB 被多次实例化，那么后面的实例会使用原来的数值进行操作。

公共变量

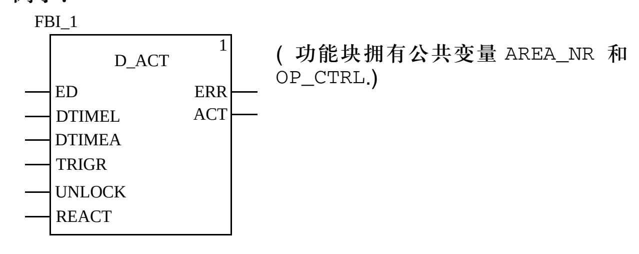
除了输入 / 输出，一些功能块还提供了公共变量。这些变量把统计数值 (不受进程影响的数值) 传递给功能块。它们用来为功能块设定参数。

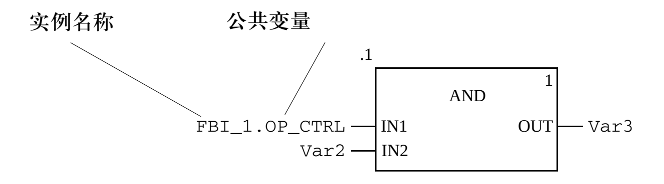
公共变量是 IEC 61131-3 的补充内容。

对公共变量的赋值是通过它们的初始值来完成的。

公共变量通过功能块的实例名称和公共变量的名称读出。

例子：





编程注意事项

在编程的过程中应该注意以下事项：

- 至少要有一个布尔输入链接了左电源母线，才能编辑 FFB。如果 FFB 没有布尔输入，就必须使用 FFB 的 EN 输入。如果 FFB 需要被有条件执行，那么布尔输入可以通过触点或者其他 FFB 进行预先链接 (请同时参见 *EN 和 ENO， 305 页*)。
- 如果 EN 输入没有链接到左电源母线，那么它必须被禁止，否则 FFB 永远也不会被处理。
- 布尔输入和输出可以取反。
- 在使用 VAR_IN_OUT 变量的时候，会用到特殊的条件 (参见 *VAR_IN_OUT 变量， 305 页*)。
- 功能块 /DFB 实例可以多次调用 (请同时参见 *多重功能块实例调用， 304 页*)。

多重功能块实例调用

功能块 /DFB 实例可以多次调用；相比之下，来自通信 EFB 和功能块 /DFB 的带有一个 ANY 输出，而不带有 ANY 输入的实例，则只能调用一次。

也可以多次调用同一个功能块 /DFB 实例，比如下面的情况：

- 如果功能块 /DFB 没有内部数值，或者在后面的处理中不需要用到它。
  在这种情况下，因为功能块 /DFB 的代码只加载一次，所以通过多次调用同一个功能块 /DFB 实例，内存会被保存起来。接下来功能块 /DFB 会像一个 “功能” 那样被处理。
- 如果功能块 /DFB 带有一个内部数值，并且该数值会影响到各个程序段，比如说，计数器的数值应该在程序的各个部分增加。
  在这种情况下，调用同一个功能块 /DFB 时，不必为将来在程序的其他部分所进行的处理保存当前结果。

EN 和 ENO

在所有 FFB 中都可以使用一个 EN 输入和一个 ENO 输出。
如果在调用 FFB 的时候， EN 的数值等于 “0”，由 FFB 所定义的算法就不会执行， ENO 会被置为 “0”。
如果在调用 FFB 的时候， EN 的数值等于 “1”，由 FFB 所定义的算法会执行。在成功执行这些算法以后， ENO 会置为 “1”。如果在执行这些算法的时候发生了错误， ENO 会置为 “0”。
如果 ENO 置为 “0” (因为 EN=0 或者执行过程中发生错误),

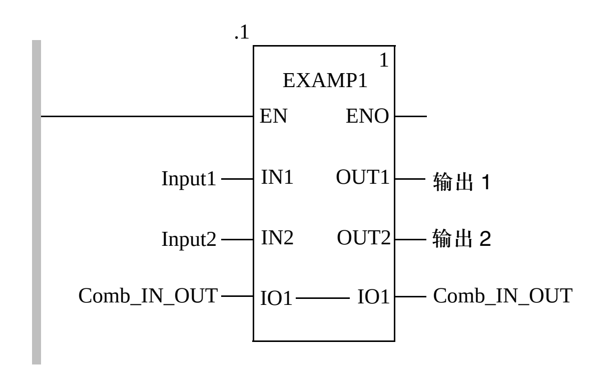
- 功能块输出会保持上一个周期的状态,
- 功能和功能程序输出置为 “0”。

> 注意：如果 EN 输入没有链接到左电源母线，那么它必须被禁止，否则 FFB 永远也不会被处理。

FFB 的输出结果，在不带 EN/ENO 或者带有 EN=1 时，是一样的。

VAR_IN_OUT 变量

FFB 经常用来在一个输入处读入值 (输入变量)，处理该变量，并用同一个变量经过修改的数值 (输出变量) 输出。
这种特殊类型的输入 / 输出变量也称为 VAR_IN_OUT 变量。
在输入和输出变量之间的链接通过 FFB. VAR_IN_OUT 变量中的一条线表示出来。



在使用带有 VAR_IN_OUT 变量的 FFB 时，需要注意以下特殊方面：

- 所有 VAR_IN_OUT 输入都必须分配一个变量。
- 不能将图形链接连到 VAR_IN_OUT 输入 / 输出。
- 不能把任何字面值或者常数连到 VAR_IN_OUT 输入 / 输出。
- 不能在 VAR_IN_OUT 输入 / 输出上使用任何取反操作。
- 必须为 VAR_IN_OUT 输入和 VAR_IN_OUT 输出连接同一个变量 / 变量成分。

控制元素

介绍

控制元素用来在 FBD 代码段中执行跳转，以及从一个子程序 (SRx) 或者导出功能块 (DFB) 返回到主程序。
控制元素占据一个单元。

控制元素

以下为可用的控制元素：

名称	演示	描述
跳转	NEXT ⟶	如果左链接的状态是 1，就会产生一个到标记的跳转 (在当前代码段)。 如果要生成一个无条件跳转，需要把跳转对象直接放置到左电源母线上。 如果要生成一个有条件跳转，需要把一个跳转对象放置到一系列触点的尾端。
标号	LABEL:	标号 (跳转目标) 以文本表示，其末尾有一个冒号。 这个文本最多可以有 32 个字符，在整个代码段中必须是唯一的。文本必须符合常规命名惯例。 跳转标记只能直接放置在电源母线的第一个单元内。 **注意：**跳转标记不能 “穿越”网络，也就是说，任何对象都不能放置在从跳转标记到代码段右边母线之间。这对跳转链接和 FFB 链接也同样适用。
返回	< RETURN >	每一个子程序和 DFB(导出功能块) 在被处理完毕以后都会退出，也就是说，返回到被调用的主程序。 如果子程序 /DFB 提早离开，可以通过返回对象强制使其返回主程序。 如果左链接的状态是 1，会从子程序或者 DFB (导出功能块) 返回到主程序。 返回对象只能用于 DFB 或者 SR 子程序。它们不能用在主程序内。 如果要生成一个有条件返回，需要把一个返回对象放置在一系列触点的尾端。

操作和比较功能块

介绍

除了在 IEC 61131-3 中定义的功能块以外，还有其他一些可以用来执行 ST 指令 (参见*指令，425 页*) 和 ST 表达式 (参见*表达式，425 页*)，以及进行简单比较操作的功能块。这些功能块只能用于 LD 编程语言。

对象

以下为可用的对象：

名称	演示	描述
操作功能块	OPERATE RES := <FCT_NAME>(...)	当左链接的状态是 1 时，在功能块中的 ST 指令会执行。 除了控制指令以外，所有 ST 指令 (参见*指令，425 页*) 都是允许的： ● (RETURN, ● JUMP, IF, ● CASE, ● FOR ● 其他) 对于操作功能块来说，左链接的状态被传送到右链接 (不管 ST 指令的结果如何)。 一个功能块最多可以包含 4096 个字符。如果不能显示所有字符，那么字符顺延的起始处后面会带有省略号 (...)。 一个操作功能块占据 1 行和 4 列。 例子： In1 指令 1; In2 指令 2; Out1 在这个例子中，如果 In1=1，指令 1 会被执行。如果 In1=1 并且 In2=1，指令 2 会被执行 (指令 1 的结果对指令 2 的执行没有意义)。如果 In1=1 并且 In2=1， Out1 会变为 1 (指令 1 和指令 2 的结果对 Out1 的状态没有意义)。

名称	演示	描述
水平比较 功能块	COMPARE <obj1 OP obj2>	水平比较功能块用来在 ST 编程语言中执行一个比较表达式 (<，>，<=，>=，=，<>) 。(注意：使用 ST 表达式也可以实现同样的功能 (参见*表达式，425 页*)。) 如果左链接的状态是 1，比较的结果是 1，右链接的状态就是 1。 一个水平比较功能块最多可以包含 4096 个字符。如果不能显示所有字符，那么字符顺序的起始处后面会带有省略号 (...)。 一个水平比较功能块占据 1 行和 2 列。 例子： In1 比较 1;; In2 比较 2; Out1 在这个例子中，如果 `In1=1`，比较 1 会被执行。如果 `In1=1`，`In2=1`，比较 1 的结果 =1，比较 2 会被执行。如果 `In1=1`，`In2=1`，比较 1 的结果 =1 并且比较 2 的结果 =1，那么 `Out1` 会变为 1 。

链接

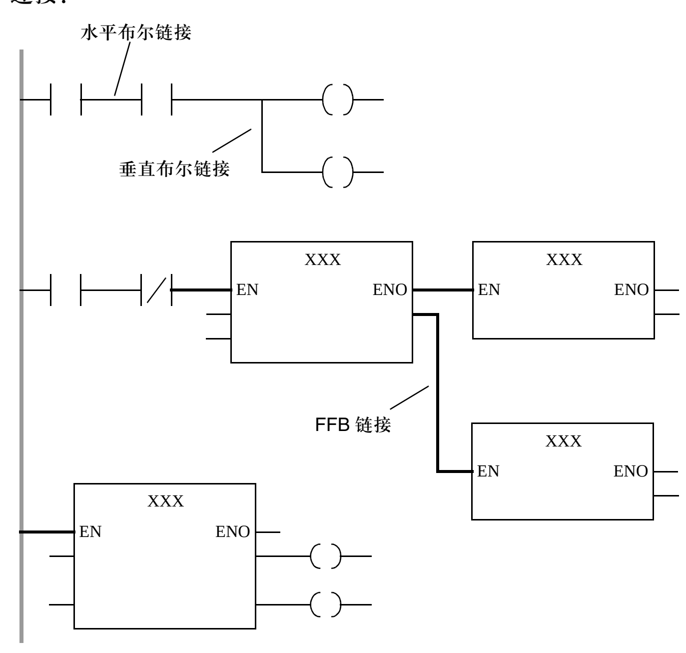
描述

链接是在 LD 对象 (触点，线圈和 FFB 等) 之间建立的连接。

链接有两种：

- 布尔链接
  把布尔对象 (触点，线圈) 彼此相连，组成一个或多个代码段。
  布尔链接也分为若干种：
  - 水平布尔链接
    水平布尔链接可连接串行的触点和线圈
  - 垂直布尔链接
    垂直布尔链接可连接并行的触点和线圈
- FFB 链接
  FFB 连接是水平和垂直代码段的组合，它能够把 FFB 输入 / 输出和其他对象连接起来。

连接：



常规编程注意事项

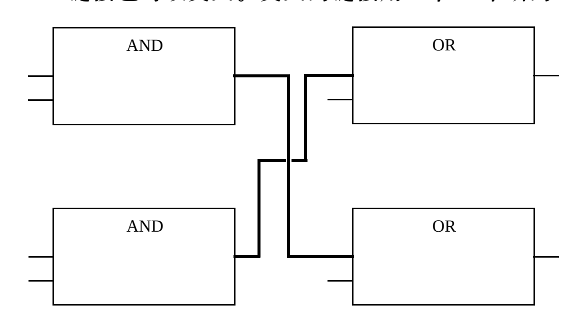
在常规编程过程中，需要注意以下方面：
- 需要链接的输入 / 输出数据类型必须是一样的。
- 带变量长度的参数 (比如 ANY_ARRAY_INT) 之间的链接是不允许的。
- 若干个链接可以连到同一个输出 (一个触点，一个线圈或者一个 FFB 输出的右侧)。不过，一个输入 (一个触点，一个线圈或者一个 FFB 输出的左侧) 只能与一个链接相连。
- 未链接的触点，线圈和 FFB 输入在缺省情况下被指定为 “0”。
- 不能使用链接来创建环路，因为在这种情况下，在代码段中无法准确地确定执行顺序。要创建环路，必须使用实际参数 (参见*设置循环，290 页*)。

编写布尔链接的注意事项

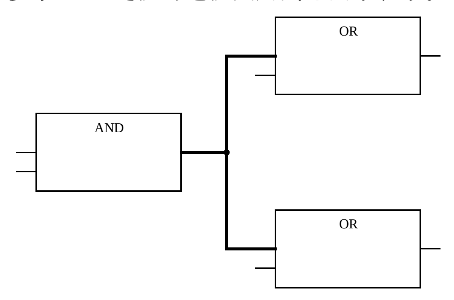
编写布尔链接的注意事项：
- 不允许布尔链接和其他对象发生重叠。
- 布尔链接的信号流 (电源流) 是自左而右的。所以不允许向后的链接。
- 如果两个布尔链接发生交叉，链接会被自动地连接起来。因为布尔链接不能交叉，所以链接不会以特殊的方式被说明。

编写 FFB 链接的注意事项

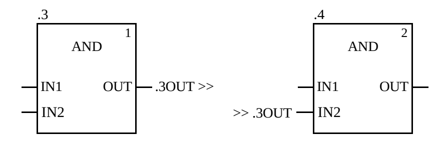
编写 FFB 链接的注意事项：
- FFB 链接至少有一侧要连到一个 FFB 输入或输出。
- 为了把 FFB 链接和布尔链接区分开来， FFB 链接用一根两倍粗的线来表示。
- 在 FFB 链接中的信号流 (电源流) 不管从哪个方向产生，它总是从 FFB 输出到 FFB 输入的。所以，向后的链接是允许的。
- 只有 FFB 输入和 FFB 输出可以互相链接。不能同时链接多个 FFB 输出。这表示在 LD 中不能使用 FFB 链接来创建 OR 连接。
- FFB 链接可以和其他对象重叠。
- FFB 链接也可以交叉。交叉的链接用一个 “中断的”链接来表示。



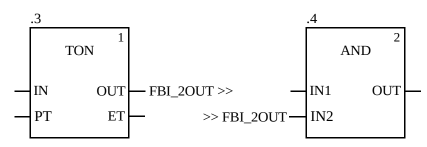
- 多个 FFB 链接的连接点用实心圆来表示。



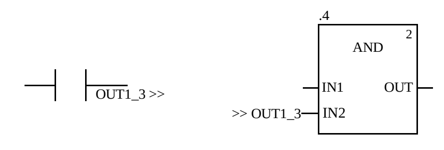
- 为了避免链接之间出现交叉，也可以用连接器来表示链接。
  连接的源和目标都带有一个名称，这些名称在代码段内是唯一的。
  根据用于连接的源对象的类型，连接器名称具有如下结构：
  - 对于功能：在连接源使用 “功能计数器 / 形式参数”



  - 对于功能块：在连接源使用 “实例名称 / 形式参数”



  - 对于触点：“OUT1_ 序号”



垂直链接

“垂直链接”是比较特殊的。垂直链接充当一个逻辑 OR。通过这种 OR 链接，用户可以使用 32 个输入 (触点) 和 64 个输出 (线圈，链接)。

文本对象

介绍

在梯形图 (LD) 中，文本可以用文本对象来定位。这些文本对象的大小取决于文本的长度。对象尺寸由文本大小决定，它可以沿垂直和水平方向扩展，占用更多的网格单元。文本对象不能与其他对象重叠。

执行顺序和信号流

网络执行顺序

以下规则适用于网络执行顺序：
- 对代码段的执行，是基于每个对象链接的自上而下，逐个网络地完成的。
- 不能使用链接来创建环路，因为在这种情况下，无法准确地确定执行顺序。要创建环路，必须使用实际参数 (参见*设置循环， 315 页*)。
- 链接到左电源母线的网络的执行顺序是由图形顺序来确定的 (自上而下)，从网络连到左电源母线的对象开始。如果有控制元素的作用，那么这条规则就不适用了。
- 在开始处理后面的网络之前，先要结束前面网络的处理。
- 所有网络元素都应该等到它全部的输入状态处理完毕以后，才能进行处理。
- 只有当一个网络的所有输出都被处理完毕，该网络的处理才算结束。如果网络包含多个控制元素，这条规则同样适用。

在网络内的信号流

对于一个网络 (梯线) 内的信号流，有如下适用规则：
- 用于布尔链接的信号流是：
  - 对水平布尔链接为从左至右
  - 对垂直布尔链接为自上而下
- FFB 链接的信号流不管在哪个方向上产生，都是从 FFB 输出到 FFB 输入的。
- 只有当与其输入相链接的所有元素(FFB输出等)被处理完毕以后，才能处理FFB。
- 与同一个 FFB 的各种输出相链接的 FFB 的执行顺序应该自上而下。
- 对象的执行顺序不受它们在网络中的位置的影响。
- FFB 的执行顺序由 FFB 的执行号码给出。

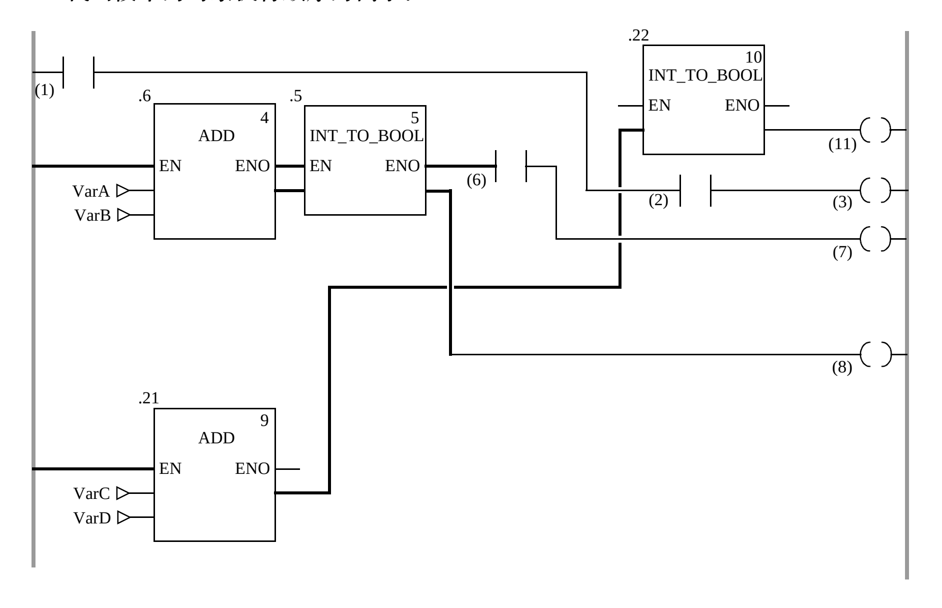
优先级

在一个代码段内定义的信号流的优先级：

优先级	规则	描述
1	链接	在一个 LD 代码段内的信号流，链接具有最高的优先级
2	逐个网络	在开始处理其他网络之前，要完全结束本网络的处理。
3	输出顺序	同一个功能块的输出或者到垂直链接的输出要自上而下进行处理。
4	逐个梯级	具有最低的优先级。从链接到左电源母线的网络的执行顺序，是由图形的顺序 (自上而下) 来决定的，信号流从左电源母线开始 (只有在其他规则都不适用的时候才使用此规则)。

例子

LD 代码段中的对象执行顺序的例子：



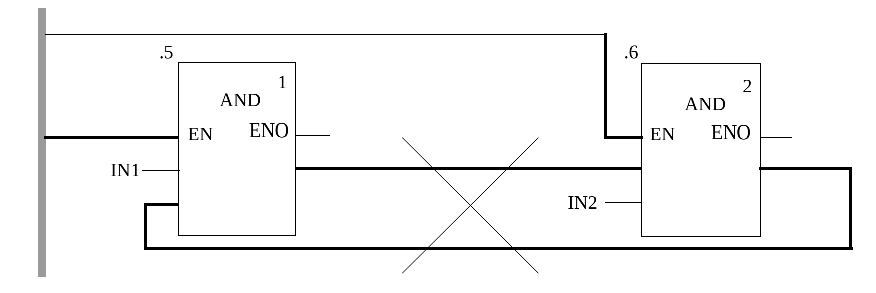
注意：触点和线圈的执行号码没有显示出来。它们只在图解中给出，以便于使用者能够更好地把握相关内容。

设置循环

不允许的环路

不能只通过链接对环路进行配置，因为无法明确说明信号流 (一个 FFB 的输出是下一个 FFB 的输入，这个 FFB 的输出又是第一个 FFB 的输入)。

不允许通过链接配置的环路：



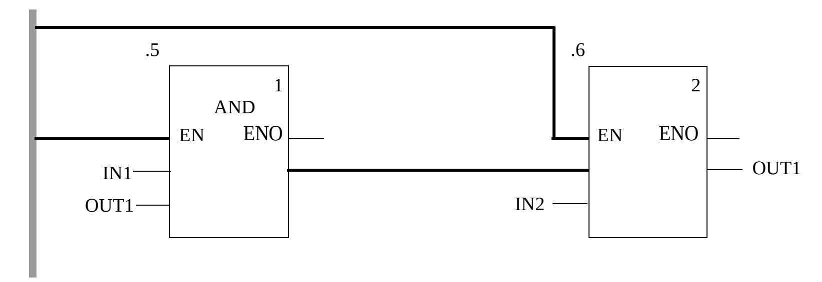
通过一个实际参数生成环路

此类逻辑必须通过反馈变量解决，以便能够确定信号流。

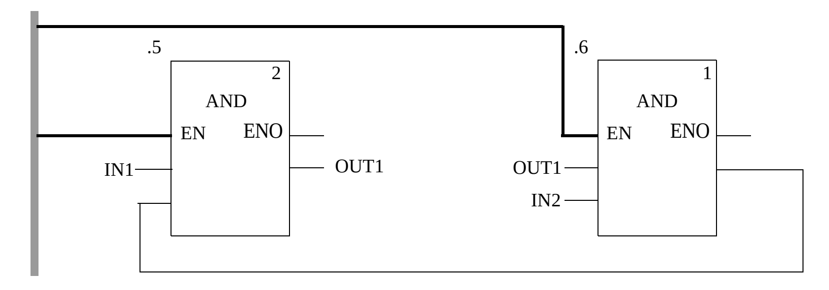
必须对反馈变量进行初始化。初始值用在逻辑的第一次执行中。在它们执行完毕以后，初始值就会被实际值所取代。

请注意用于两种功能块的两种不同的执行顺序 (在实例名称后面的括号中的数值)。

通过一个实际参数生成的环路：第 1 类



通过一个实际参数生成的环路：第 2 类



更改执行顺序

介绍

网络的执行顺序和一个网络中的对象的执行顺序，是通过一系列规则进行定义的 (参见 *执行顺序， 313 页*)。

在某些情况下，用户应该更改由系统推荐的执行顺序。

定义 / 更改网络执行顺序的功能程序如下所示：

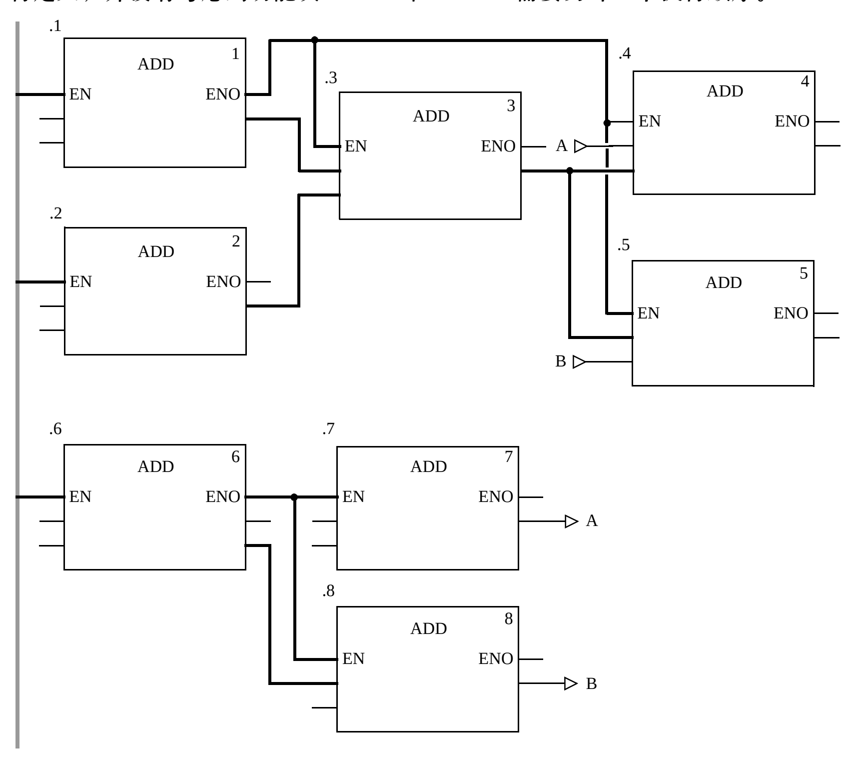
- 使用链接，而不是实际参数
- 网络位置

定义 / 更改网络执行顺序的功能程序如下所示：

- 对象的位置

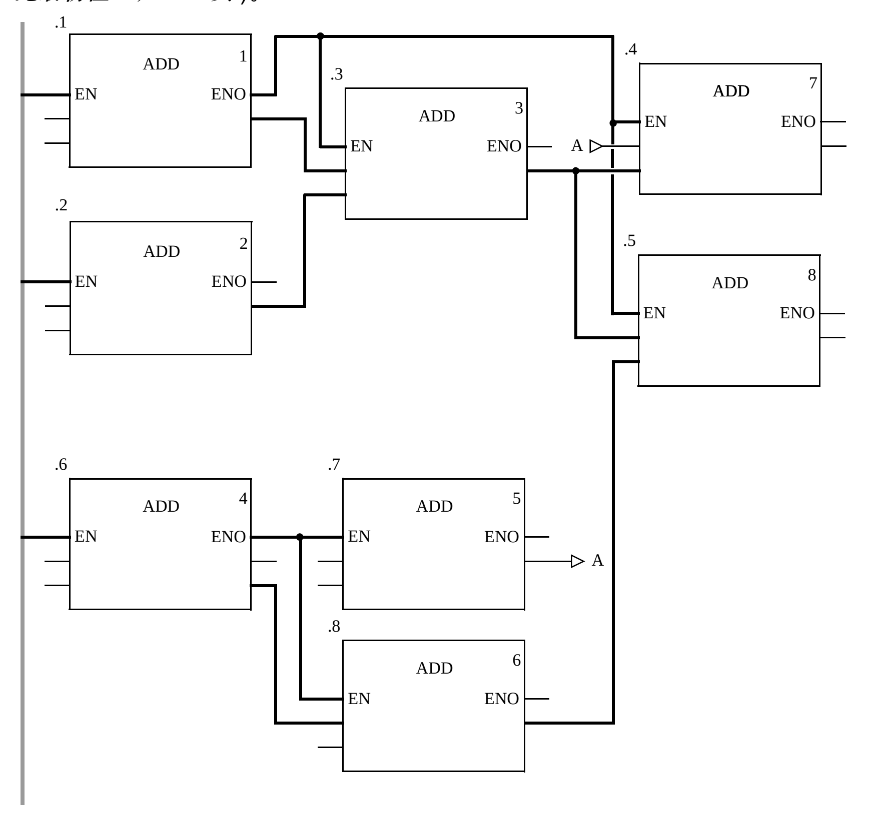
最初位置

在下面的演示中给出了两个网络，它们的执行顺序仅通过它们在代码段中的位置进行定义，并没有考虑到功能块 0.4/0.5 和 0.7/0.8 需要另外一个执行顺序。



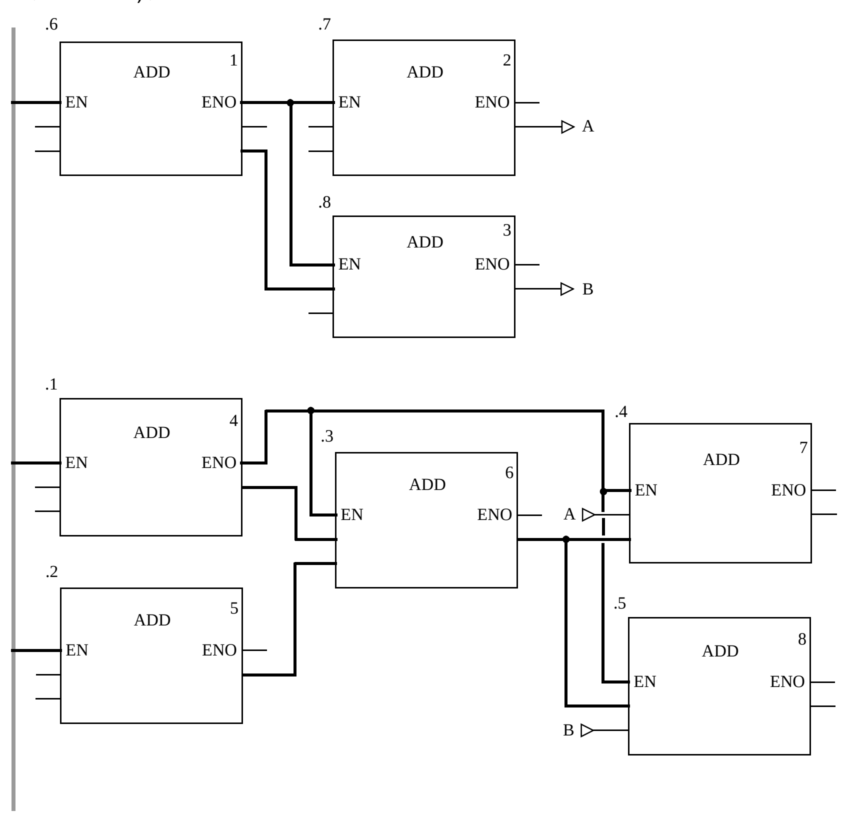
使用链接，而非实际参数

通过使用一个链接，而不是一个变量，两个网络可以以正确的顺序运行 (请同时参见*最初位置， 316 页*)。



网络位置

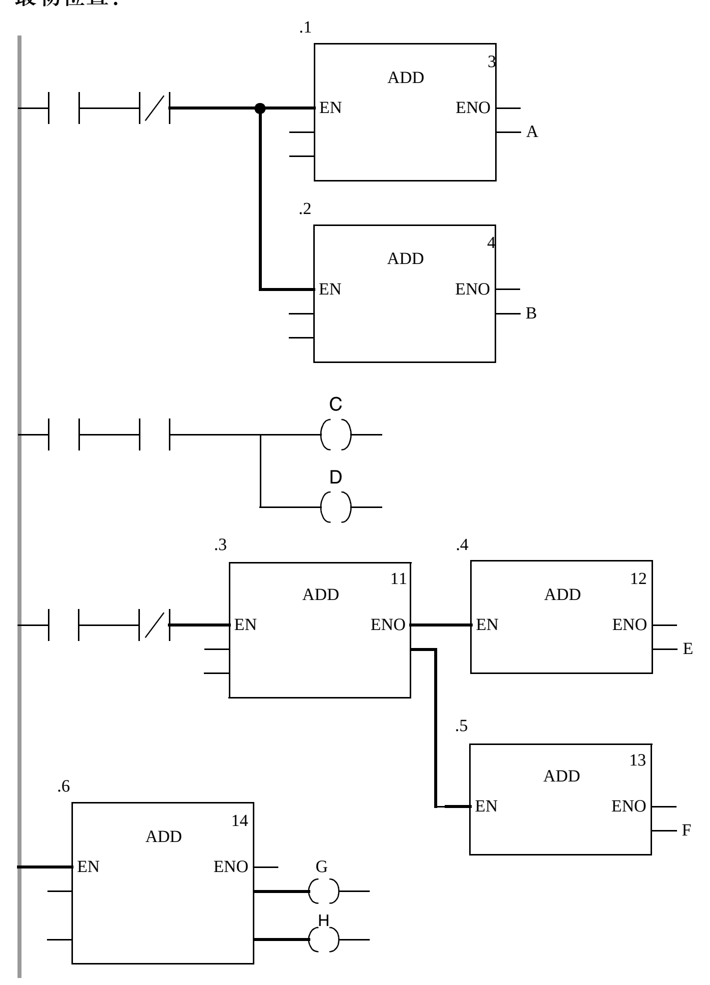
可以通过在代码段中更改网络的位置来获得正确的执行顺序 (请同时参见*最初位置， 316 页*)。



对象的定位

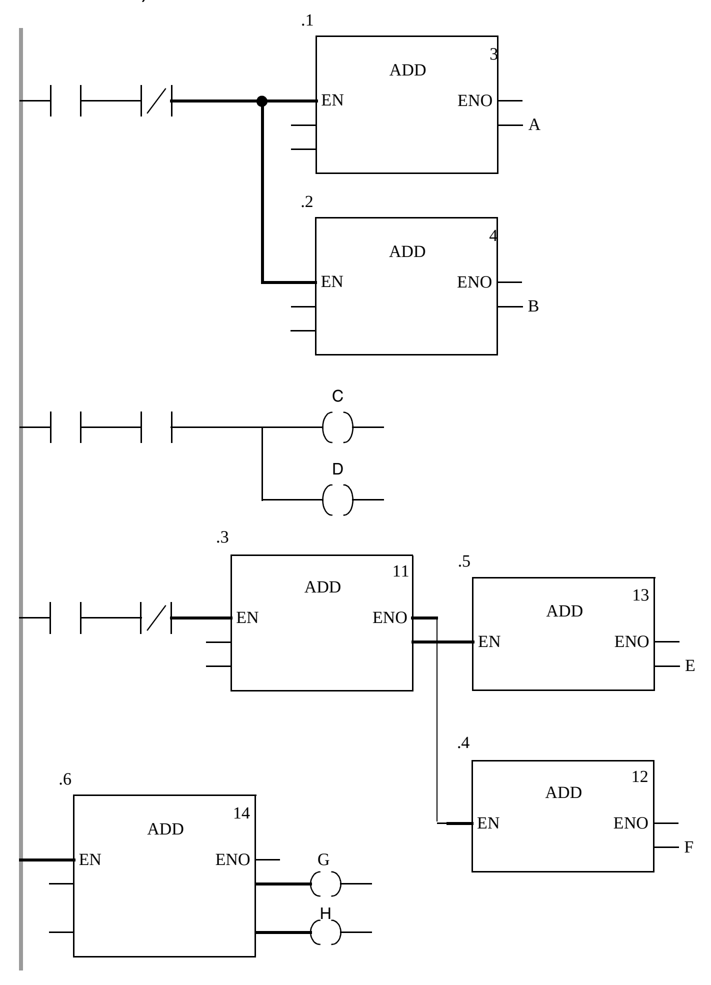
如果若干个输入 (触点 / 线圈的左链接， FFB 输入) 与 “要调用”的对象的同一个输出 (触点 / 线圈的右链接， FFB 输出) 进行链接，那么对象的位置只对执行顺序有影响 (请同时参见*最初位置，316 页*)。

最初位置：



在第一个网络中，功能块的位置 `.1` 和 `.2` 被交换了。在这种情况下 (两个功能块的输入具有公共的来源)，两个功能块的执行顺序也被交换了 (自上而下进行处理)。在第二个网络中交换线圈 C 和 D 时，情况也是一样的。

在第三个网络中，功能块的位置 .4 and .5 被交换了。在这种情况下 (两个功能块的输入具有不同的来源)，两个功能块的执行顺序没有被交换 (以调入功能块输出的顺序进行处理)。在最后一个网络中交换线圈 G 和 H 时，情况也是一样的。



SFC 顺序功能图

13

介绍

概述

本章介绍了遵循 IEC 611311 的 SFC 顺序功能图。

本章内容

本章包含以下各节：

13.1 关于 SFC 顺序功能图的常规信息

介绍

概述

本节给出了 SFC 顺序功能图的综述

本节内容

本节包含以下主题：

关于 SFC 顺序功能图的常规信息

介绍

本节描述了遵循 IEC 61131-3SFC 的顺序功能图语言 (顺序功能图)。

顺序控制器的结构

遵循 IEC 的顺序控制是在 Unity Pro 中通过 SFC 代码段 (顶级)，转换代码段和动作代码段创建的。
这些 SFC 代码段只能用于项目的主任务中。 SFC 代码段不能用在其他任务或者 DFB 中。
每一个 SFC 代码段都一个或多个 SFC 网络 (顺序串)。

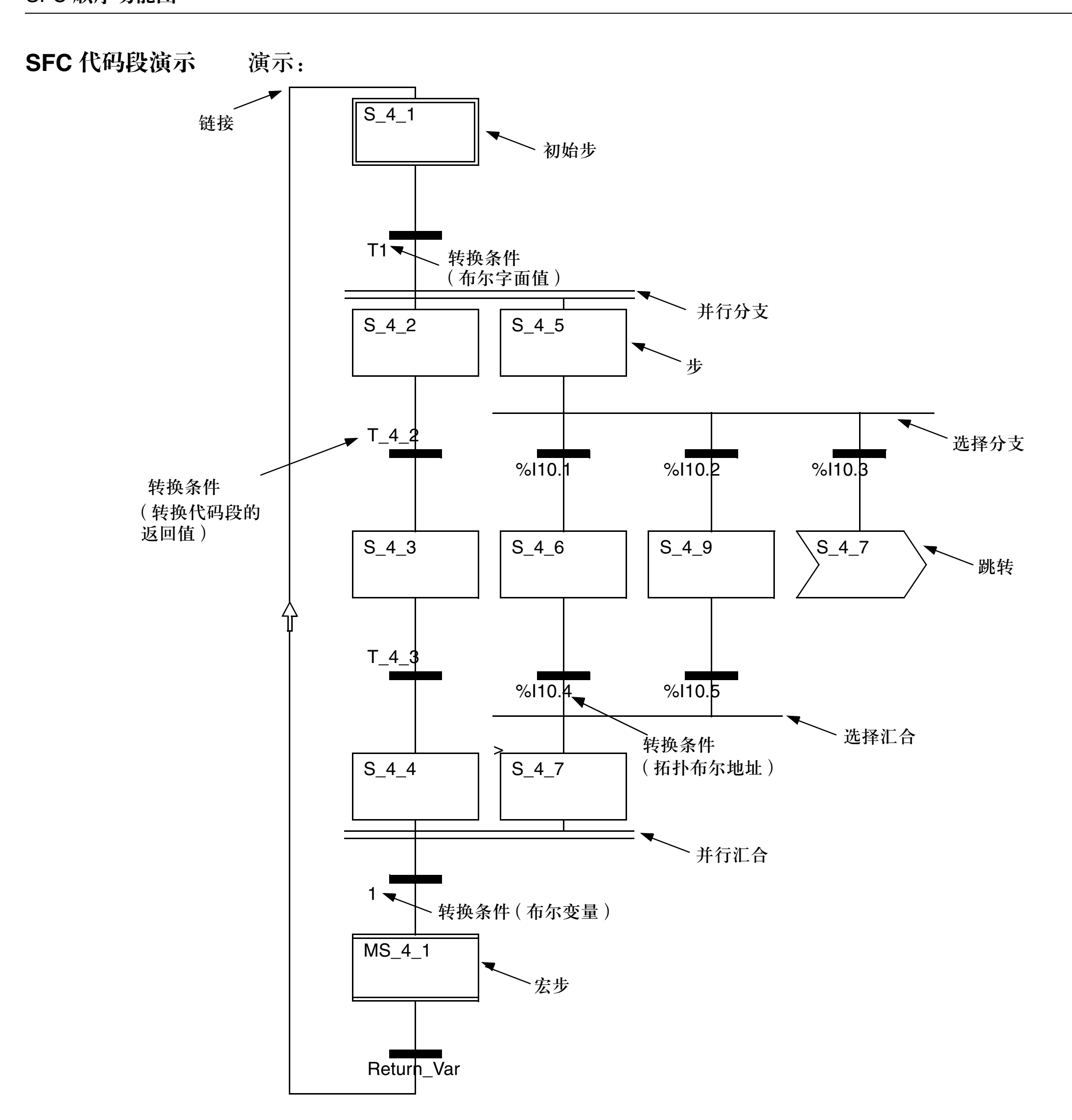
对象

SFC 代码段为创建程序提供了如下对象：

- 步 (参见*步，329 页*)
- 宏步 (嵌入式子步串) (参见*宏步和宏代码段，332 页*)
- 转换 (转换条件) (参见*转换，344 页*)
- 跳转 (参见*跳转， 348 页*)
- 链接 (参见*链接， 349 页*)
- 选择分支 (参见*选择分支和选择汇合， 351 页*)
- 选择汇合 (参见*选择分支和选择汇合， 351 页*)
- 并行分支 (参见*并行分支和并行汇合， 352 页*)
- 并行汇合 (参见*并行分支和并行汇合， 352 页*)

对于带有文本对象 (相关的主题*文本对象， 353 页*) 的代码段逻辑，可以给出注释。

SFC 代码段演示 演示：


链接
S_4_1
初始步
T1
转换条件
（布尔字面值）
并行分支
S_4_2
S_4_5
步
选择分支
T_4_2
%I10.1
%I10.2
%I10.3
转换条件
（转换代码段的
返回值）
S_4_3
S_4_6
S_4_9
S_4_7
跳转
T_4_3
%I10.4
%I10.5
选择汇合
转换条件
（拓扑布尔地址）
S_4_4
S_4_7
并行汇合
1
转换条件（布尔变量）
MS_4_1
宏步
Return_Var

SFC 代码段的结构

ASFC 代码段是一个 “状态机器”，也就是说，状态是通过有效步和转换的切换 / 更改行为来创建的。步和转换通过定向链接相连。两个步永远不能直接链接，它们必须被一个转换分隔开。有效信号状态进程沿着定向链接发生，它通过变换转换来触发。串进程的方向与定向链接相同，从前任步的低侧进入后续步的高侧。分支则从左向右进行处理。

每个步可能不含有动作，也可能含有若干动作。每个转换都需要用到一个转换条件。在串中的最后一个转换总是和串中的另外一个步相连 (通过一个图形链接或者跳转符号)，从而能够创建一个闭合电路。这样一来步串就可以以循环的方式执行。

SFCCHART_STATE 变量

如果创建了一个 SFC 代码段，那么它就会被自动分配一个数据类型变量 SFCCHART_STATE. 被创建的变量总是代表相关 SFC 代码段的名字。

这个变量分配给 SFC 的控制功能块，来控制 SFC 代码段。

令牌规则

SFC 网络的行为在很大程度上受所选令牌数量，也就是说有效步的数量的影响。

通过使用一个令牌 (单令牌)，可以实现明确的行为。 (在并行分支中，每一个分支都有一个作为单令牌的有效令牌 [步])。这是遵循 IEC 61131-3 标准的一个步串。

带有由用户定义的最大有效步 (多令牌) 的步串提升了编程自由度。它减少 / 消除了在实现明确性和非模块化过程中的限制，但它必须由用户提供担保。带有多令牌的步串不遵循 IEC 61131-3 标准。

代码段大小

- SFC 代码段含有一个单页面窗口。
- 这个窗口有一个逻辑网格，其中有 200 行和 32 列。
- 步，转换和跳转分别需要用到一个单元。
- 分支和链接不需要用到单元，它们被插入到相关步或者转换的单元中去。
- 每个 SFC 代码段最多可以容纳 1024 个步 (包括它们所有的宏代码段)。
- 每个SFC代码段最多可以有100个有效步(多令牌)(包括它们所有的宏代码段)。
- 每个 SFC 代码段最多 64 个步 (多令牌) 可以用手动同时设置。
- 每个 SFC 步最多可以分配 20 个动作。
- 宏步的嵌入深度，例如：宏步中的宏步，为 8 级。

遵循的 IEC 标准

关于 SFC 顺序功能图语言所遵循的 IEC 标准，请参见 *“遵循的 IEC 标准”*。

链接规则

链接规则

下面的表格说明了哪些对象输出和对象输入可以互相链接。 .

从 ... 的对象输出	到 ... 的对象输入
步	转换
	选择分支
	并行汇合
转换	步
	跳转
	并行分支
	选择汇合
选择分支	转换
	并行汇合 (仅限于多令牌 (参见*多令牌*， *365 页*))
选择汇合	步
	跳转
	并行分支
	选择汇合
并行分支	步
	跳转
	选择汇合 (仅限于多令牌 (参见*多令牌*， *365 页*))
并行汇合	转换
	选择分支 (仅限于多令牌 (参见*多令牌*， *365 页*))
	选择汇合

13.2 步和宏步

介绍

概述

本节描述了 SFC 顺序功能图的步和宏步对象。

本节内容

本节包含以下主题：

步

步类型

有以下步类型：

类型	演示	描述
“正常”步	S_1_2	如果前面的步变为无效 (一个可能被定义的延迟必须通过)，并且上游的转换被满足，那么相应的步就会变为有效。通常情况下，如果一个可能被定义的延迟通过，并且下游的转换被满足，那么相应的步就会变为无效。对于一个并行汇合来说，前面的所有步都必须满足这些条件。 每一个步可能不含有动作，也可能含有若干个动作。不含有动作的步被称为等待步。
初始步	S_1_1	一个顺序串的初始状态由初始步来描绘。在对项目或者顺序串进行了初始化以后，初始步就是有效的了。通常情况下，初始步不分配任何动作。 在单令牌的情况下 (符合 IEC 61131-3)，每一个顺序串只允许带有一个初始步。 在多令牌的情况下，初始步的数量是可定义的 (0 到 100)。
宏步	MS_1_1	参见*宏步，332 页*
输入步	MS_1_1_IN	参见*输入步，332 页*
输出步	MS_1_1_OUT	参见*输出步，333 页*

步名称

在创建一个步的时候，会分配一个推荐号码。这个推荐号码有如下结构：S_i_j，其中 i 是代码段的 (内部) 当前号码， j 是当前代码段 (内部) 当前步的号码。您可以出于方便的需要而更改这个推荐号码。步名称 (最多 32 个字符) 在整个项目中必须是唯一的，也就是说，其他任何步，变量或者代码段都不能具有同样的名称。名称不区分大小写。步名称必须符合标准的命名惯例。

步时间

每一个步都可以分配一个最小的监管时间，一个最大的监管时间，以及一个延迟时间：

- **最小监管时间**
  最小监管时间设定步在通常情况下保持有效的最短时间。如果在该时间结束以前步变为无效，就会产生一个出错信息。在动态模式下，会用一个有色的轮廓线 (黄色) 在步对象周围把错误标记出来。
  如果没有输出最小监管时间或者输入的最小监管时间为 0，就不会进行步监管操作。
  错误状态会保持不变，直到步重新变为有效。
- **最大监管时间**
  最大监管时间定义步在通常情况下保持有效的最长时间。如果在该时间结束以后步仍然有效，就会产生一个出错信息。在动态模式下，会用一个有色的轮廓线 (粉色) 在步对象周围把错误标记出来。
  如果没有输出最大监管时间或者输入的最大监管时间为 0，就不会进行步监管操作。
  错误状态会保持不变，直到步重新变为无效。
  延迟时间
  延迟时间 (步保留时间) 设定步必须保持有效的最小时间。

注意：上面所定义的时间只适用于步，不适用于分配的动作。它们可以定义属于自己的时间。

设定步时间

下面的规则用来定义 / 确定这些时间：

延迟时间 < 最小监管时间 < 最大监管时间

把定义的数值分配给一个步，有两种方法：

- 作为一个持续字面值
- 使用数据结构 SFCSTEP_TIMES

SFCSTEP_TIMES 变量

每一个步都可以隐式地被分配有一个数据类型 SFCSTEP_TIMES 的变量。这种数据结构的元素可以被读出和写入 (读 / 写)。

这种数据结构和其他数据结构的操作方式是一样的，也就是说，它们可以用于变量声明，进而能够访问整个数据结构 (比如作为 FFB 参数)。

该数据结构的结构：

元素名称	数据类型	描述
"VarName".delay	TIME	延迟时间
"VarName".min	TIME	最小监管时间
"VarName".max	TIME	最大监管时间

SFCSTEP_STATE 变量

每一个步都可以隐式地被分配有一个数据类型 SFCSTEP_STATE 的变量。这个步变量具有分配的步的名称。这种数据结构只能被读出 (只读)。

这种数据结构不能用于变量声明。所以，用户不能访问整个数据结构 (比如作为 FFB 参数)。

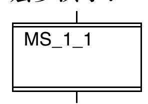
该数据结构的结构：

元素名称	数据类型	描述
"StepName".t	TIME	在该步中的当前停留时间。如果步处于无效状态，这个元素的值会被保留，直到步重新被激活。
"StepName".x	BOOL	1：步有效 0：步无效
"StepName".tminErr	BOOL	这个元素是 IEC 61131-3 的补充内容。 1：最小监管时间下溢 0：最小监管时间没有下溢 在以下情况下，该元素会被自动复位： ● 如果步重新被激活 ● 如果顺序控制被复位 ● 如果命令按钮复位时间错误**(Reset Time Error)** 被激活
"StepName".tmaxErr	BOOL	这个元素是 IEC 61131-3 的补充内容。 1：最大监管时间溢出 0：最大监管时间没有溢出 在以下情况下，该元素会被自动复位： ● 如果该步被退出 ● 如果顺序控制被复位 ● 如果命令按钮复位时间错误**(Reset Time Error)** 被激活

宏步和宏步代码段

宏步

宏步用来调用宏代码段，进而用来组织顺序控制的层次结构。
宏步演示：

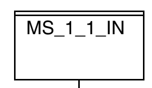


宏步有如下属性：
- 宏步可以放在 “顺序控制”代码段和宏代码段中。
- 宏步的数量没有限制。
- 嵌套深度，也就是在宏步中的宏步最多可以有 8 级。
- 每一个宏步都会隐式地被分配有一个数据类型 SFCSTEP_STATE 的变量，参见 SFCSTEP_STATE 变量， 331 页。
- 宏步可以被分配有一个数据类型 SFCSTEP_TIMES 的变量，参见 SFCSTEP_TIMES 变量， 331 页。
- 不能将动作分配给宏步。
- 每一个宏步在分配的宏代码段中都可以被顺序串所代替。

宏步是 IEC 61131-3 的补充内容，必须被明确激活。

输入步

每一个宏代码段都从一个输入步开始。
输入步演示：



输入步有如下属性：
- 输入步只能放在宏代码段中。
- 对于每一个宏代码段，只能放置一个输入步。
- 每一个输入步都会隐式地被分配有一个数据类型 SFCSTEP_STATE 的变量，参见 SFCSTEP_STATE 变量， 331 页。
- 输入步可以被分配有一个数据类型 SFCSTEP_TIMES 的变量，参见 SFCSTEP_TIMES 变量， 331 页。
- 输入步可以被分配有动作。

输出步

每一个宏代码段都以一个输出步结尾。
输出步演示：

MS_1_1_OUT

输出步有如下属性：
- 输出步只能放在宏代码段中。
- 对于每一个宏代码段，只能放置一个输出步。
- 输出步不能分配有动作。
- 输出步只能分配延迟时间，不能分配监管时间，参见*步时间， 330 页。*

宏代码段

宏代码段含有一个顺序串，其元素与 “顺序控制”代码段基本相同 (比如步，初始步，宏步，转换，分支，汇合，等等)。
另外，每一个宏代码段在开始部分还有一个输入步，在结尾部分有一个输出步。
每一个宏步都可以在用具有宏代码段的顺序串代替。
所以，宏代码段可以包含 0 个， 1 个或多个初始步，参见*步类型， 329。*
- 单令牌
  - 0 个初始步
    用于宏代码段，如果在较高或较低的代码段中已经有一个初始步。
  - 1 个初始步
    用于宏代码段，如果在较高或较低的代码段中没有初始步。
- 多令牌
  每个代码段最多可以放置 100 个初始步 (包括它们所有的宏代码段)。

使用宏代码段：

宏代码段的名称与被调用的宏步的名称是一样的。如果宏步的名称被改变，那么相应的宏代码段名称也会自动更改。
一个宏代码段只能使用一次。

宏步处理

宏步处理：

阶段	描述
1	如果前面的转换条件为 TRUE，宏步会被激活。与此同时，在宏代码段中的输入步会被激活。
2	宏代码段的顺序串会被处理。 只要在宏代码段中有一个步处于有效状态，宏步就会保持有效。
3	如果宏代码段的输出步有效，那么宏步后面的转换就可以使能。
4	如果输出步激活，而且后面的转换条件是激活的，转换条件为 TRUE，那么宏步就会变为无效。与此同时，在宏代码段中的输出步也会无效。

步名称

在创建一个步的时候，它会被分配有一个推荐号码。
推荐号码的含义：

步类型	推荐的号码	描述
宏步	MS_i_j	MS = 宏步 i = 当前代码段的 (内部) 段 (顺序) 号 j = 当前代码段的 (内部) 当前 (顺序) 步号
输入步	MS_k_l_IN	MS = 宏步 k = 调用代码段的 (内部) 段 (顺序) 号 l = 调用代码段的 (内部) 当前 (顺序) 步号 IN = 输入步
输出步	MS_k_l_OUT	MS = 宏步 k = 调用代码段的 (内部) 段号 (顺序) l = 调用代码段的 (内部) 当前 (顺序) 步号 OUT = 输出步
“正常”步 (在一个宏代码段内)	S_k_m	S = 步 k = 调用代码段的 (内部) 段 (顺序) 号码 m = 调用代码段的 (内部) 当前 (顺序) 步号

为了方便使用，您可以更改推荐的号码。步名称 (对宏步最多为 28 个字符，对步名称最多为 32 个字符) 在整个项目中必须是唯一的，也就是说，其他任何步，变量或者代码段 (除了分配给宏步的宏代码段名称) 等都不能具有同样的名称。名称不区分大小写。步名称必须符合标准命名惯例。
如果宏步的名称被更改，那么相应的宏代码段和其中的步的名称也会自动更改。
比如说，如果 MS_1_1 被重新命名为 MyStep，那么在宏代码段中的步名称就会被重新命名为 MyStep_IN, MyStep_1, ..., MyStep_n, MyStep_OUT.

13.3 动作和动作代码段

介绍

概述 本节描述了 SFC 顺序功能图的动作和动作代码段。

本节内容 本节包含以下主题：

动作

介绍

动作有以下属性：

- 动作可以是编程语言FDB，LD，IL或者ST的一个布尔变量 (动作变量(参见*动作变量， 338 页*)) 或者一个代码段 (动作代码段 (参见*动作代码段， 339 页*))。
- 一个步可以不分配动作，也可以分配若干个动作。一个未分配动作的步具有等待功能，也就是说，它会等待到分配的转换完成为止。
- 如果多个动作同时被分配给一个步，它们会按照在动作列表栏中的排列顺序进行处理。

例外：带有标识符 (参见*可用标识符， 340 页*) `P1` 的动作不受它们在动作列表栏中的位置的影响，它们总是被首先处理，而带有标识符 `P0` 的动作总是被最后处理。

- 动作控制通过标识符来表达 (参见*标识符， 340 页*)
- 每个步最多可以分配 20 个动作。
- 分配给动作的的动作变量也可以用于来自其他步的动作。
- 动作变量也可以用于项目中其他代码段的读或写操作 (多重分配)。
- 分配了带有持续时间的标识符的动作只能激活一次。
- 只有布尔变量 / 地址或者多元素变量中的布尔元素可以作为动作变量使用。
- 动作具有唯一的名称。
- 动作的名称是动作变量的名称，或者动作代码段的名称。

动作变量

以下是可用的动作变量：
- BOOL 数据类型的地址

动作可以通过一个地址分配给硬件输出。在这种情况下，动作可以用作转换的使能信号，另一个代码段的输入信号，以及硬件的输出信号。
- BOOL 数据类型的简单变量或者多元素变量中的元素

在另外一个代码段的变量的帮助下，动作可以用作输入信号。
  - 非定位型变量

  通过非定位型变量，动作可以用作转换的使能信号以及另一个代码段的输入信号。
  - 定位型变量

  通过定位型变量，动作可以用作转换的使能信号， 另一个代码段的输入信号，以及硬件的输出信号。

动作名称

如果一个地址或者变量用作动作，那么它的名称 (比如 %Q10.4，Variable1) 就用作动作名称。

如果一个动作代码段用作动作，那么代码段名称就被用作动作名称。

动作名称 (最多为 32 个字符) 在整个项目中必须是唯一的，也就是说，任何其他转换，变量或者代码段等都不能具有同样的名称。名称不区分大小写。动作名称必须符合标准的命名惯例。

动作代码段

介绍

每个动作都可以创建一个动作代码段。这个代码段包含动作的逻辑，并且与动作自动链接。

动作代码段名称

动作代码段的名称与分配的动作的名称总是一致的，参见*动作名称， 338 页*。

编程语言

FBD， LD， IL 和 ST 是动作代码段可用的编程语言。

动作代码段属性

动作代码段有如下属性：
- 动作代码段的输出数量没有限制。
- 只有激活了多令牌操作以后，才能在动作代码段中进行子程序调用。
  注意：被调用的子程序不受顺序串控制器的影响，也就是说
  - 分配给调用动作代码段的标识符不影响子程序，
  - 如果调用的步没激活，子程序仍然保持有效。
- 在动作代码段中不能使用诊断功能，诊断功能块或者诊断功能程序。
- 动作代码段的网络数量没有限制。
- 动作代码段隶属于定义它们的SFC代码段，并且可以在该SFC代码段内被分配任意数量的动作 (包括它们所有的宏代码段)。
- 被分配了带有持续时间的标识符的动作代码段只能被激活一次。
- 动作代码段隶属于定义它们的 SFC 代码段，如果相应的 SFC 代码段被删除，那么该 SFC 代码段的所有动作代码段都会自动删除。
- 动作代码段也可以只从动作中调用。

标识符

介绍

与一个步相链接的每一个动作都必须有一个标识符，该标识符定义动作的控制。

可用的标识符

以下为可用的标识符：

标识符	描述	描述
N / None	未存储	如果步有效，动作就为 1 ；如果步无效，动作就为 0。
R	具有最高优先级的复位	把另外一个步中使用标识符 S 置位的动作被复位。与此同时，也可以避免任何其他动作被激活。 **注意：**标识符会自动地声明为无缓冲。这意味着它的数值在停止和冷启动之后会复位为 0，比如说当电源电压断开 / 接通的时候。如果需要用到缓冲的输出，请使用标准功能块库中的 RS 或者 SR 功能块。

<table>
<tr><th>标识符</th><th>描述</th><th>描述</th></tr>
<tr><td>S</td><td>设定的
(保存的)</td><td>当相关的步无效时，这个设定的动作仍然保持有效。只有当动作在当前顺序串的另外一个步中通过标识符 R 被重置时，它才会变为无效。
<table><tr><th>代码段 _1</th><th>代码段 2</th></tr><tr><td>S_1_1
S A1
S_1_2
S_1_3</td><td>IF S2.x= true THEN
    A1:= false;
END_IF;</td></tr></table>
A1
S_1_1
S_1_2
S_1_3

注意：标识符会自动声明为无缓冲。这意味着它的数值在停止和冷重启之后会复位为 0，比如说当电源电压断开 / 闭合的时候。如果需要用到缓冲的输出，请使用标准功能块库中的 RS 或者 SR 功能块。</td></tr>
<tr><td>L</td><td>限定时间</td><td>如果步是有效的，那么动作也是有效的。在人工定义的持续时间过去以后，即使步仍然有效，动作也会返回到 0。如果步无效，动作也为 0。
注意：对于这个标识符，必须定义一个额外的数据类型 TIME 的持续时间。</td></tr>
</table>

标识符	描述	描述
D	延迟	如果步有效，内部计时器会启动，在人工设定的持续时间结束以后，动作会变为 1。如果在此之后步变为无效，那么动作也会变为无效。如果在内部时间到达之前，步变为无效，那么动作不会变为有效。 **注意：**对于这个标识符，必须定义一个额外的数据类型 TIME 的持续时间。
P	脉冲	如果步变为有效，动作会变为 1，并且，不管步是否保持有效状态，动作的 1 状态都会保持一个程序周期。
DS	延迟并保存	如果步变为有效，内部计时器会启动，在人工设定的延迟时间到达后，动作变为有效。当在另外一个步中使用标识符 R 进行复位操作时，动作会重新变为无效。 如果在内部时间到达之前，步变为无效，那么动作不会变为有效。 **注意：**对于这个标识符，必须定义一个额外的数据类型 TIME 的持续时间。
P1	脉冲 (上升沿)	如果步变为有效 (0->1- 边沿)，动作会变为 1，并且，不管步是否保持有效状态，动作的 1 状态都会保持一个程序周期。 **注意：**带有标识符 P1 的动作不受它们在动作列表栏中的位置的影响，它们总是被最先处理，参见*动作， 337 页*。
P0	脉冲 (下降沿)	如果步变为有效 (1->0- 边沿)，动作会变为 1，并且，不管步是否保持有效状态，动作的 1 状态都会保持一个程序周期。 **注意：**带有标识符 P0 的动作不受它们在动作列表栏中的位置的影响，它们总是被最后处理，参见*动作， 337 页*。

13.4 转换和转换代码段

介绍

概述

本章描述了 SFC 顺序功能图的转换对象和转换代码段。

本节内容

本节包含以下主题：

转换

介绍

转换为一个或多个预转换步是否按一个或多个连续步沿着相应的链接进行提供了条件。

转换条件

每一个转换都会分配一个 BOOL 数据类型的转换条件。下面是可用的转换条件：
- 一个地址（输入或者输出）
- 一个变量（输入或者输出）
- 一个字面值或者
- 一个转换代码段（参见*转换代码段， 346 页*）

转换条件的类型决定了名称被放置的位置：

转换条件	名称的位置
• 地址 • 变量	Trans_Variable
• 字面值	1
• 转换代码段	Trans_Section

转换名称

如果一个地址或变量用作转换条件，那么转换名称会通过该名称进行定义（比如 %I10.4， Variable1）。
如果一个转换代码段用作转换条件，那么代码段名称会用作转换名称。
转换名称（最多为 32 个字符）在整个项目中必须是唯一的，也就是说，其他任何转换，变量或者代码段都不能具有同样的名称。名称不区分大小写。转换名称必须符合标准的命名惯例。

激活一个转换

如果转换前面的相邻步是有效的，那么这个转换就被激活。如果一个转换前面的相邻步不是有效的，通常我们不去分析该转换。

注意：如果没有定义转换条件，转换永远也无法变为有效。

触发一个转换

如果一个转换被激活，并且相关的转换条件得到了满足，那么这个转换就会被触发。

如果一个转换被触发，那么与该转换相链接的，位于它前方的相邻步都会失效 (复位)，而后位于它后方的相邻步则会被激活。

转换的触发时间

从理论上来说，转换触发时间 (转换时间) 可以尽可能的短，但是不能为零。转换触发时间至少应该维持一个程序周期。

转换代码段

介绍

每个转换都可以创建一个转换代码段。它是一个包含转换逻辑条件的代码段，自动与转换相链接。

转换代码段的名称

转换代码段的名称与被分配的转换的名称总是一致的，参见*转换名称， 344 页*。

编程语言

FBD， LD， IL 和 ST 是转换代码段可用的编程语言。

转换代码段的推荐网络：

语言	推荐的网络	描述
FBD	AND IN1 OUT — T1 IN2	推荐的网络包含一个带有 2 个输入的 AND 功能块，相关的输出与具有转换代码段名称的变量相链接。推荐的功能块可以被链接，也可以根据需要删除掉。
LD	T1 —( )—	推荐的网络包含一个线圈，该线圈与一个具有转换代码段名称的变量相链接。推荐的线圈可以被链接，也可以根据需要删除掉。

IL	-	推荐的网络是空的。
		其内容只能通过布尔逻辑来创建。在这个输出 (转换变量) 上的逻辑结果分配是自动完成的，也就是说，内存分配 ST 是不允许的。
		例子：
		`LD A`
		`AND B`

语言	推荐的网络	描述

ST	-	推荐的网络是空的。
		其内容只能通过布尔逻辑，以一个 (嵌套) 表达式的方式来创建。这个输出 (转换变量) 上的逻辑结果分配是自动完成的，也就是说，赋值指令：= 是不允许的。表达式不是通过分号 (;) 来结束的。
		例子：
		`A AND B`
		or
		`A AND (WORD_TO_BOOL (B))`

转换代码段的属性

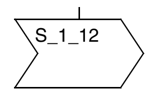
转换代码段有以下属性：
- 转换代码段只有一个输出 (转换变量)，它的数据类型是 BOOL。这些变量的名称与转换代码段的名称相同。
- 转换变量在书面形式中只能使用一次。
- 用户可以在项目中的任何位置读转换变量。
- 用户只能使用功能，不能使用功能块或功能程序。
- 在 LD 中只能使用一个线圈。
- 只有一个网络，也就是说，所有使用的功能都以直接或者间接的方式彼此链接。
- 转换代码段只能使用一次。
- 转换代码段属于定义它们的 SFC 代码段。如果相应的 SFC 代码段被删除，那么这个 SFC 代码段的所有转换代码段也会自动删除。
- 转换代码段只能被转换进行调用。

13.5 跳转

跳转

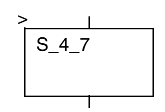
常规信息

跳转用来指出不用表示其全长的定向链接。
跳转的演示：



跳转的属性

跳转具有以下属性：
- 多个跳转可以拥有同一个目标步。
- 根据 IEC 61131-3，跳转不能进入一个并行顺序 (参见*并行分支和并行汇合，352 页*) 或者跳出一个并行顺序。
- 如果要再次使用它，必须明确将它激活。
- 对于跳转来说，在顺序跳转 (参见*顺序跳转，357 页*) 和顺序环路 (参见*顺序环路，358 页*) 之间存在着差别。
- 跳转目标以跳转目标符号 (>) 来表示。



跳转名称

跳转实际上没有自己的名称。在跳转符号中给出的是目标步 (跳转目标) 的名称。

13.6 链接

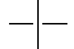
链接

介绍

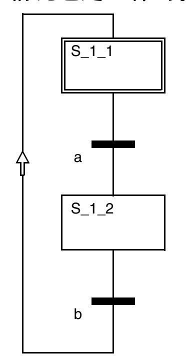
链接用来在步和转换之间建立连接。

链接的属性

链接具有以下属性：
- 在同一类型的对象 (步和步，转换和转换， 等等) 之间不能建立链接。
- 可以在以下各项之间建立链接：
  - 未链接的对象输出和
  - 未链接的或者链接的步输入
    (也就是说可以链接多重输入)
- 不能让链接和其他 SFC 对象 (步，转换，跳转， 等等) 发生重叠
- 不能让链接之间发生重叠
- 链接之间可以交叉，这种情况通过一个 “断开的”链接来表示：



- 链接包含垂直和水平段
- 在一个顺序串中的标准信号流是自上而下的。不过为了创建一个环路，链接可以从下方进入一个上方的步。对从转换，并行分支或者选择汇合到一个步的链接，情况也是一样的。在这里，链接的方向会用一个箭头符号表示出来：



- 对于链接来说，在串跳转 (参见*顺序跳转，357 页*) 和串环路 (参见*顺序环路，358 页*) 之间存在着差别。

13.7 分支和汇合

介绍

概述

本节描述了 SFC 顺序功能图的分支和汇合对象。

本节内容

本节包含以下主题：

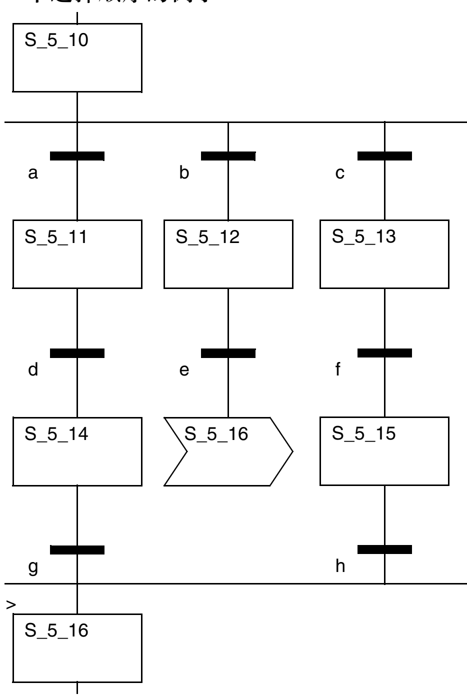
选择分支和选择汇合

介绍

通过选择分支，用户可以在 SFC 结构的控制进程中，编写有条件的分支。
通过选择分支，如果一个步的水平线后面带有多个转换，那么就有多个进程。
所有选择分支都通过选择汇合或者跳转并入一个单一的分支 (参见*跳转， 348 页*)，进行进一步处理。

选择顺序的例子

一个选择顺序的例子



选择顺序的属性

选择顺序的属性主要取决于顺序控制是在单令牌模式还是多令牌模式下操作的。
See

- 在单令牌模式下的顺序控制的属性 (参见*选择串， 356 页*)
- 在多令牌模式下的顺序控制的属性 (参见*选择串， 357 页*)

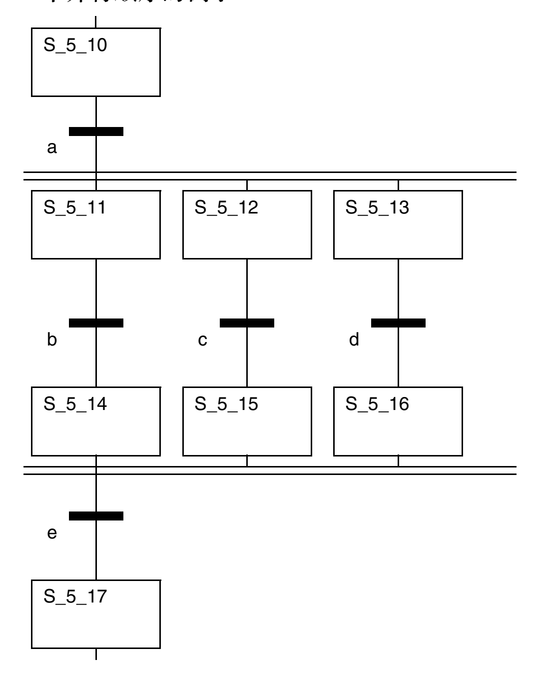
并行分支和并行汇合

介绍

通过并行分支，对一个单一的转换进行切换操作，会同时触发多个 (最多 32 个) 步 (分支)。执行的顺序是从左到右。在这次整体触发以后，每一个分支会独立处理。根据 IEC 61131-1，所有并行分支都会通过一个并行汇合集合起来。在与并行汇合相邻的所有前任步都被激活以后，在并行汇合后面的转换才会计算。
只有在多令牌操作中，才能把一个并行分支和一个选择汇合联合使用 (参见*并行串， 370 页*)。

并行顺序的例子

一个并行顺序的例子



并行顺序的属性

参见
- 在单令牌模式下的顺序控制的属性 (参见*选择串， 356 页*)
- 在多令牌模式下的顺序控制的属性 (参见*选择串， 357 页*)

13.8 文本对象

文本对象

介绍

通过 SFC 顺序功能图，可以把文本以文本对象的形式进行定位。这些文本对象的大小取决于文本的长度。文本对象至少应该有一个单元的大小，它可以根据文本的尺寸沿着水平和垂直方向扩展到其他单元中去。文本对象可以和其他 SFC 对象重叠。

13.9 单令牌

介绍

概述

本节描述了用于顺序控制的 “单令牌” 操作模式。

本节内容

本节包含以下主题：

单令牌执行顺序

描述

以下规则适用于单令牌：

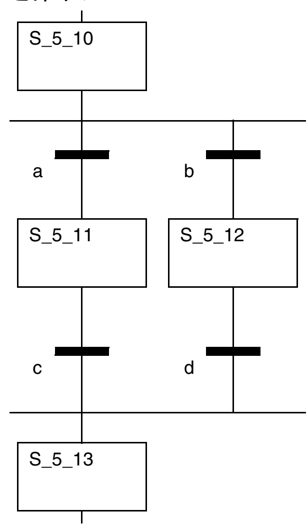
- 输出的情况由初始步来定义。顺序串只包含一个初始步。
- 在顺序串中只含有一个有效步。唯一的例外是并行分支，在并行分支中，每一个分支都有一个有效步。
- 有效信号状态的进程沿着定向链接发生，它通过转换条件来触发。串进程的方向与定向链接相同，从前任步的低侧进入后继步的高侧。分支会从左向右进行处理。
- 如果转换前面的相邻步是有效的，那么这个转换就会被激活。如果一个转换前面的相邻步不是有效的，通常我们不去分析该转换。
- 如果一个转换被激活，并且相关的转换条件得到了满足，那么这个转换就会被触发。
- 如果一个转换被触发，那么与该转换相链接的，位于它前方的相邻步都会停止 (复位)，而后位于它后方的相邻步则会被激活。
- 如果在一系列顺序步中满足了多个转换条件，那么每个周期会执行一个步。
- 步不能被非 SFC 代码段激活或者禁止。
- 可以使用宏步。
- 在选择分支中只能有一个分支是有效的。要运行的分支是由跟在选择分支后面的转换的转换条件结果决定的。如果满足了一个转换条件，那么就不再处理剩余的转换。满足条件的转换所在的分支会被激活。在此处的操作中，所有分支的优先级按照从左到右逐渐降低的顺序排列。所有选择分支都会在结尾处通过一个选择汇合或者跳转集合起来。
- 通过并行分支，对一个单一的转换进行切换操作，会同时触发多个步 (分支)。在这次整体触发以后，每一个分支会独立处理。所有并行分支都会通过一个并行汇合集合起来。不能跳转到一个并行分支，也不能从一个并行分支跳出。

选择串

选择串

根据 IEC 61131-3，对转换只能进行一次切换 (1- 关闭 -n- 选择)。要运行的分支是由跟在选择分支后面的转换的转换条件结果来决定的。分支转换按照从左到右的顺序处理。如果满足了一个转换条件，那么就不再处理其余的转换。满足条件的转换所在的分支会被激活。在此处的操作中，所有分支的优先级按照从左到右逐渐降低的顺序排列。如果没有切换任何转换，当前被设置的步会保持设定状态。

选择串：

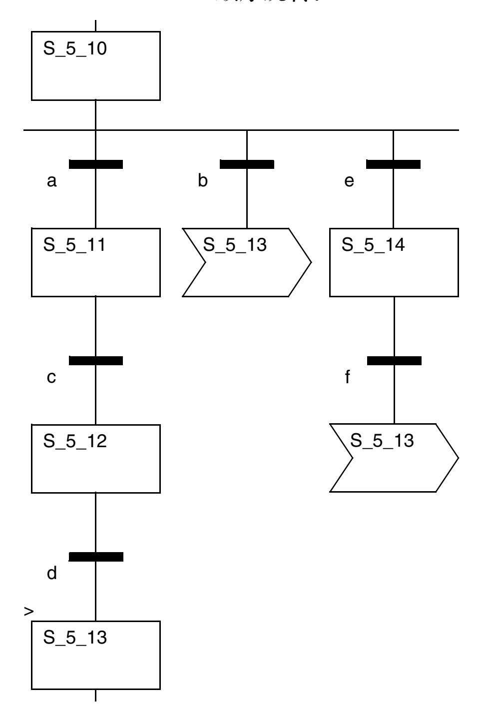


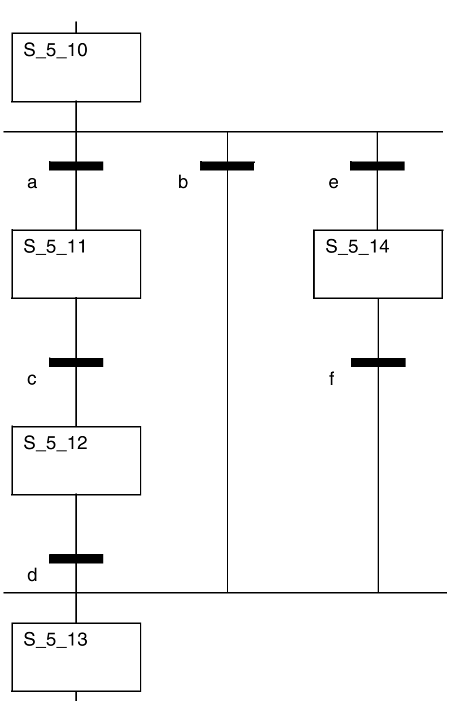
如果 ...	那么
如果 S_5_10 有效，并且转换条件 a 为真 (与 b 无关)，	那么就从 S_5_10 到 S_5_11 运行一个顺序。
如果 S_5_10 有效，并且转换条件 b 为真， a 为假，	那么就从 S_5_10 到 S_5_12 运行一个顺序。

顺序跳转和顺序环路

顺序跳转

顺序跳转是一类特殊的选择分支，可以用来略过一个顺序中的若干个步。
顺序跳转可以通过跳转或者链接来实现。
顺序跳转：





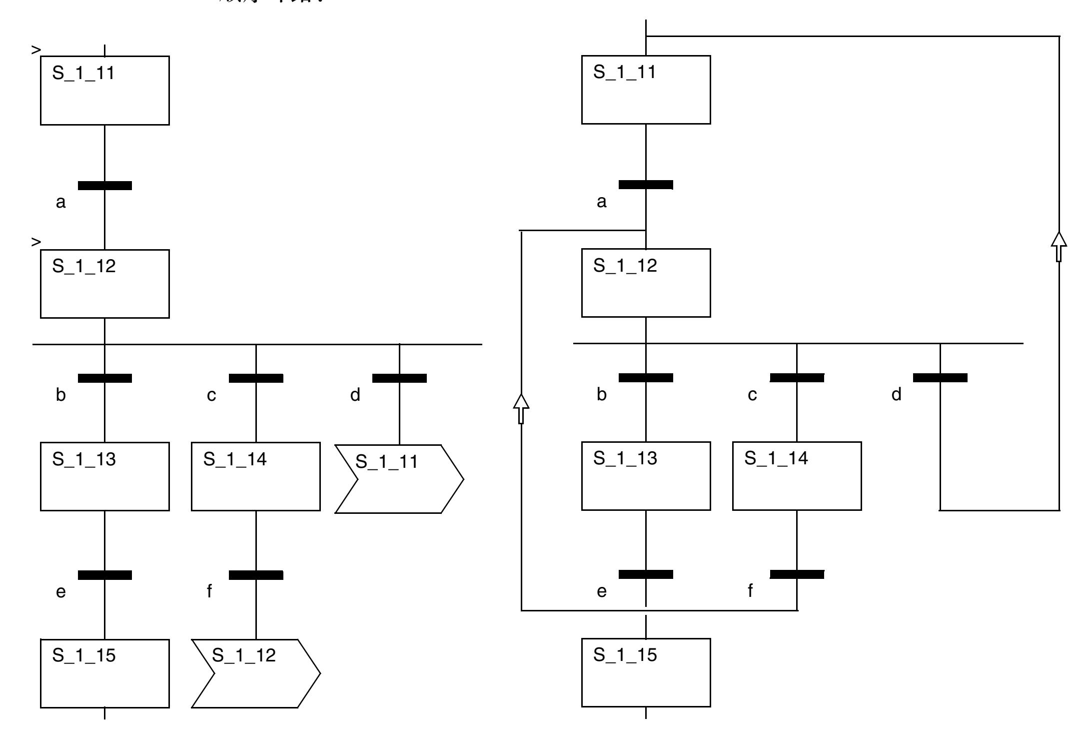
如果 ...	那么
如果转换条件 a 为真，	那么会从 S_5_10 到 S_5_11， S_5_12 和 S_5_13 运行一个顺序。
如果转换条件 b 为真，	那么会从 S_5_10 到 S_5_13 直接进行一次跳转。
如果转换条件 e 为真，	那么会从 S_5_10 到 S_5_14 和 S_5_13 运行一个顺序。

顺序环路

顺序环路是一类特殊的选择分支，通过顺序环路，一个或者多个分支可以返回到一个前任步。

顺序环路可以通过跳转或者链接来实现。

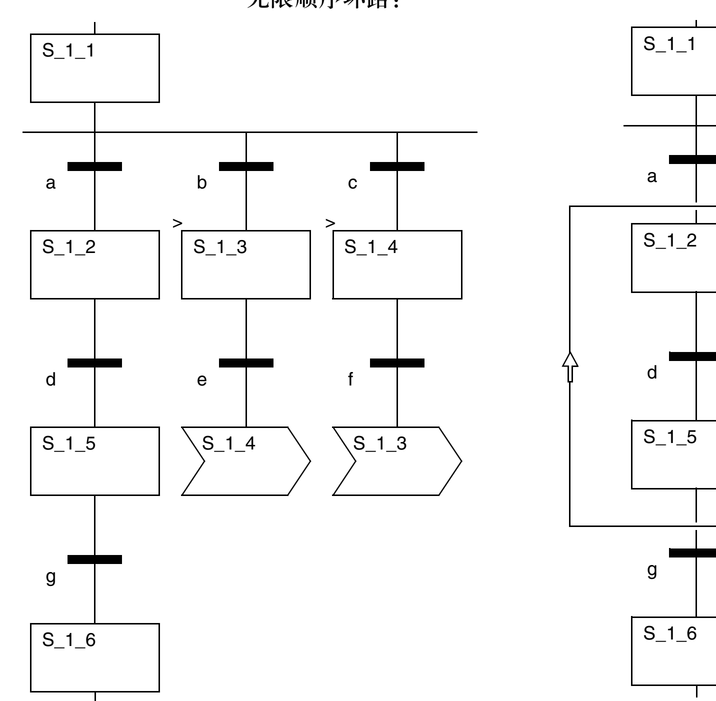
顺序环路：

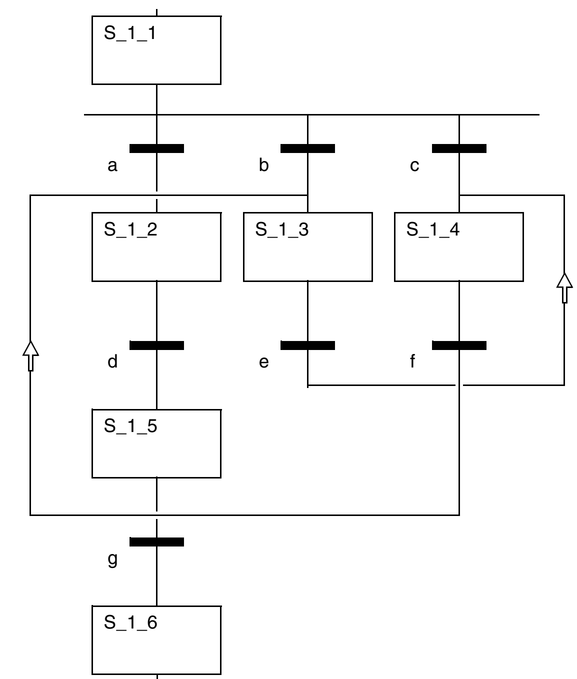


如果 ...	那么
如果转换条件 a 为真，	那么会从 S_1_11 到 S_1_12 运行一个顺序。
如果转换条件 b 为真，	那么会从 S_1_12 到 S_1_13 运行一个顺序。
如果转换条件 b 为假，并且 c 为真，	那么会从 S_1_12 到 S_1_14 运行一个顺序。
如果转换条件 f 为真，	那么会有一次跳转 S_1_14 返回到 S_1_12。
从 S_1_12 借助转换条件 c 和 f 返回到 S_1_12 的环路会一直重复下去，直到转换条件 b 为真或者 c 为假，并且 d 为真。	
如果转换条件 b 和 c 为假，并且 d 为真，	那么会有一次跳转从 S_1_12 直接返回到 S_1_11。
借助转换条件 a 和 d 从 S_1_11 到 S_1_12 并返回 S_1_11 的环路会一直重复下去，直到转换条 b 或者 c 为真。	

在选择顺序内不允许出现无限顺序环路。

无限顺序环路：





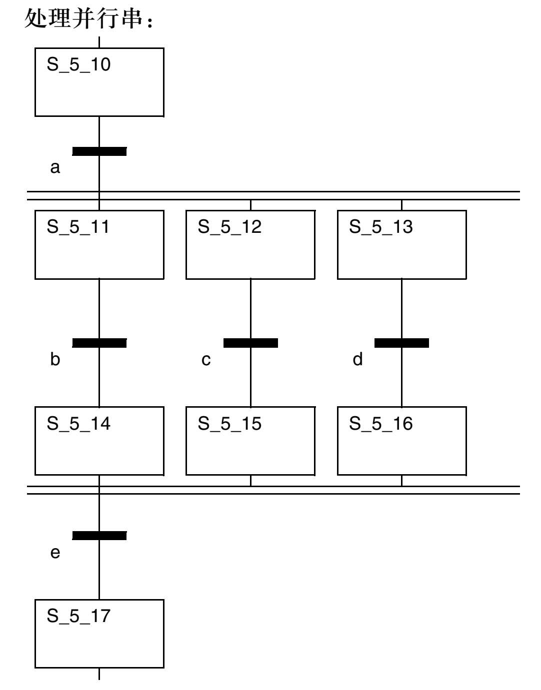
如果 ...	那么
如果转换条件 b 为真，	那么从 S_1_1 到 S_1_3 会运行一个顺序。
如果转换条件 e 为真，	那么会有一个到 S_1_4 的跳转。
如果转换条件 f 为真，	那么会有一个到 S_1_3 的跳转。
现在，环路借助转换条件 e 从 S_1_3 出发，借助转换条件 f 到达 S_1_4，并且再次跳转回到 S_1_3，从而无限循环下去。	

并行串

并行串

通过并行分支，对一个单一的转换进行切换操作，会同时触发多个 (最多 32 个) 步 (分支)。这对单令牌和多令牌同样适用。

处理并行串：

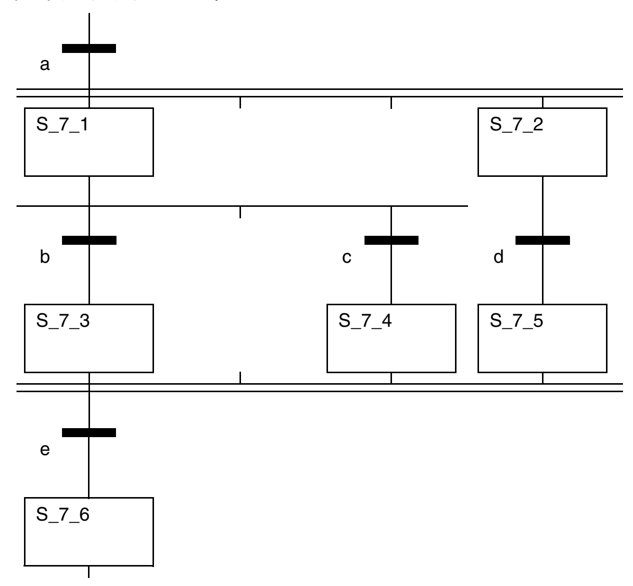


如果 ...	那么
如果 S_5_10 有效，并且隶属于公共转换的转换条件 a 也为真，	那么从 S_5_10 到 S_5_11， S_5_12 和 S_5_13 就会运行一个顺序。
如果步 S_5_11， S_5_12 和 S_5_13 有效，	那么各个串之间就会彼此独立运行。
如果 S_5_14， S_5_15 和 S_5_16 同时有效，并且隶属于公共转换的转换条件 e 也为真，	那么从 S_5_14， S_5_15 和 S_5_16 到 S_5_17 就会运行一个顺序。

在并行串中使用选择分支

如果在一个并行串中使用了单一的选择分支，那么就可能导致单令牌的串被封闭。

在并行串中使用选择分支：



如果 ...	**那么**
如果转换条件 a 为真，	那么会有一个顺序运行到 S_7_1 和 S_7_2。
如果步 S_7_1 和 S_7_2 被激活，	那么各个串之间就会彼此独立运行。
如果转换条件 d 为真，	那么会有一个顺序运行到 S_7_5.
如果转换条件 b 为真，并且 c 为假，	那么会有一个顺序运行到 S_7_3.
考虑到 S_7_3, S_7_4 和 S_7_5 与一个并行汇合相链接，S_7_6 永远不会触发，因为 S_7_3 和 S_7_4 无论如何也不能同时有效。 (或者 S_7_3 通过转换条件 b 激活，或者 S_7_4 通过转换条件 c 激活，两者不能同时发生。) 所以 S_7_3, S_7_4 和 S_7_5 也不能同时有效。串被封闭了。 在进入选择分支的时候，如果转换条件 b 为假， c 为真，那么也会出现同样的情况。	

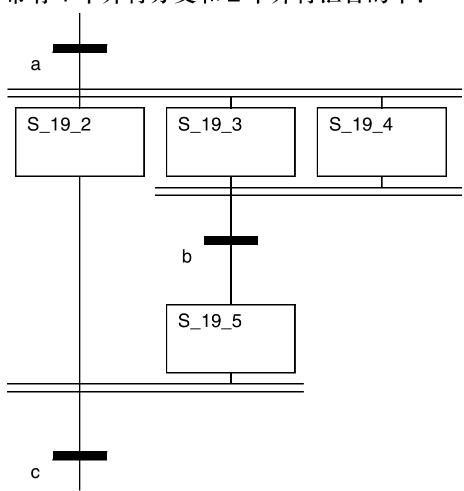
非对称并行串选择

介绍

根据 IEC 61131-3，一个并行分支必须以一个并行汇合来中止。并行分支的数量必须与并行汇合的数量相符合。

汇合数量较多的情况

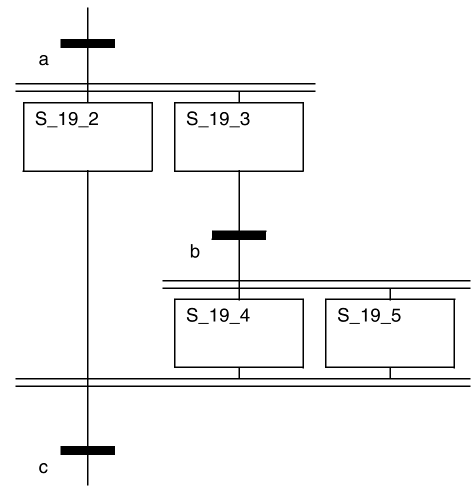
带有 1 个并行分支和 2 个并行汇合的串：



如果 ...	那么
如果转换条件 a 为真，	那么会有一个顺序运行到 S_19_2, S_19_3 和 S_19_4。
如果步 S_19_2, S_19_3 和 S_19_4 有效，	那么各个串之间就会彼此独立运行。
如果转换条件 b 为真，	那么会有一个顺序运行到 S_19_5。
如果步 S_19_2 和 S_19_5 有效，并且转换条件 c 为真，	那么就会离开并行串。

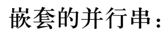
分支数量较多的情况

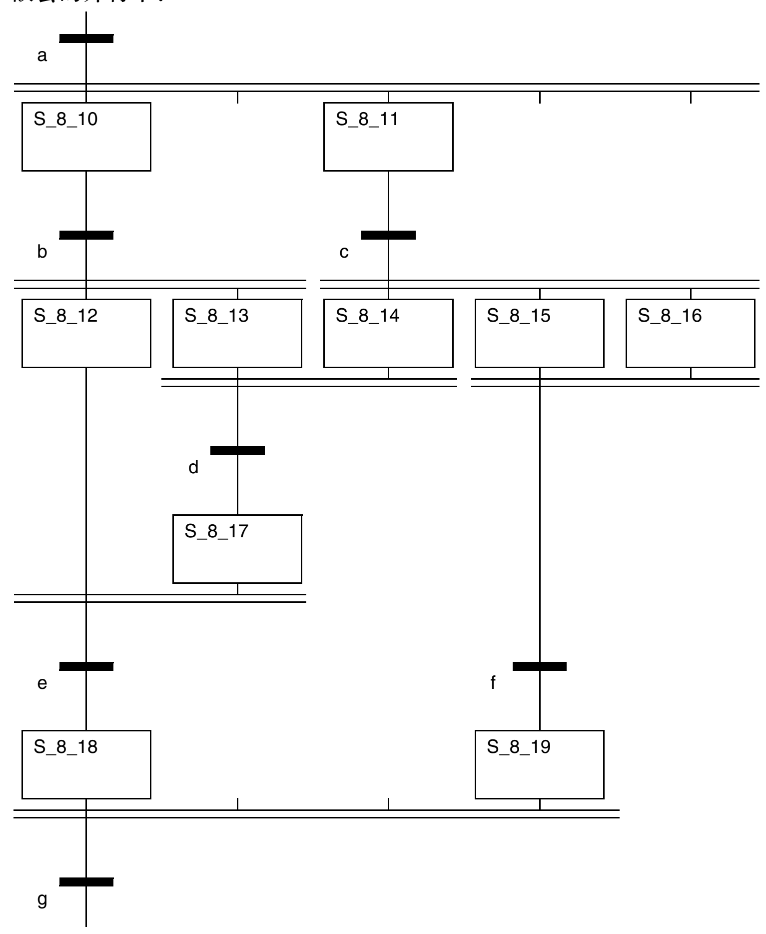
带有 2 个并行分支和 1 个并行汇合的串：



如果 ...	那么
如果转换条件 a 为真，	那么会有一个顺序运行到 S_19_2 和 S_19_3。
如果步 S_19_2 和 S_19_3 有效，	那么各个串之间就会彼此独立运行。
如果转换条件 b 为真，	那么会有一个顺序运行到 S_19_4 和 S_19_5。
如果步 S_19_4 和 S_19_5 有效，	那么各个串之间就会彼此独立运行。
如果步 S_19_2, S_19_4 和 S_19_5 有效，并且转换条件 c 为真，	那么就会离开并行串。

嵌套的并行串


嵌套的并行串：



如果 ...	那么
如果转换条件 a 为真，	那么会有一个顺序运行到 S_8_10 和 S_8_11。
如果转换条件 b 为真，	那么会有一个顺序运行到 S_8_12 和 S_8_13。
如果转换条件 c 为真，	那么会有一个顺序运行到 S_8_14, S_8_15 和 S_8_16。
如果步 S_8_13 和 S_8_14 有效，并且转换条件 d 为真，	那么会有一个顺序运行到 S_8_17。
如果步 S_8_12 和 S_8_17 有效，并且转换条件 e 为真，	那么会有一个顺序运行到 S_8_18。
...	...

13.10 多令牌

介绍

概述

本节描述了用于顺序控制的 “多令牌”操作模式。

本节内容

本节包含以下主题：

多令牌执行顺序

描述

以下规则适用于多令牌：

- 用户可以定义初始位置的一系列初始步 (0 到 100 个)。
- 在一个顺序串中，被自由定义的一系列步，可以在同一时间内同时有效。
- 有效信号状态进程沿着定向链接发生，它通过变换转换来触发。串进程的方向与定向链接一致，从前任步的低侧进入后继步的高侧。
- 如果转换前面的相邻步是有效的，那么这个转换就会被激活。如果一个转换前面的相邻步不是有效的，通常我们不去分析该转换。
- 如果一个转换被激活，并且相关的转换条件得到了满足，那么这个转换就会被触发。
- 如果一个转换被触发，那么与该转换相链接的，位于它前方的相邻步都会停止 (复位)，而后位于它后方的相邻步则会激活。
- 如果在一系列顺序步中满足了多个转换条件，那么每个周期会执行一个步。
- 步和宏步可以通过非 SFC 代码段或者用户操作激活或禁止。
- 如果对一个有效步同时进行了激活和禁止操作，那么该步会保持有效状态。
- 可以使用宏步，宏步代码段也可以包含初始步。
- 通过选择分支，可以使多个分支同时有效。要运行的分支是由跟在选择分支后面的转换条件结果决定的。分支转换是以并行的方式处理的。带有被满足的转换的分支会被激活。所有选择分支可以不必在结尾处通过一个选择汇合或者跳转集合起来。
- 如果要向并行分支内，或者从并行分支向外进行跳转操作，那么可以激活这个选项。在这种情况下，所有选择分支可以不必在结尾处通过一个选择汇合集合起来。
- 在动作代码段内可使用子程序调用。
- 多令牌通过以下的方式创建：
  - 多重初始步
  - 未中止的选择或者并行分支
  - 结合选择和并行串的跳转
  - 在一个非 SFC 代码段使用 SFC 控制功能块 `SETSTEP`，或者使用 SFC 控制指令来激活相关步
- 令牌可以通过以下列方式中止：
  - 在一个步内同时汇合两个或者多个令牌
  - 从一个非 SFC 代码段使用 SFC 控制功能块 `RESETSTEP`，或者使用 SFC 控制指令来停止相关步

选择串

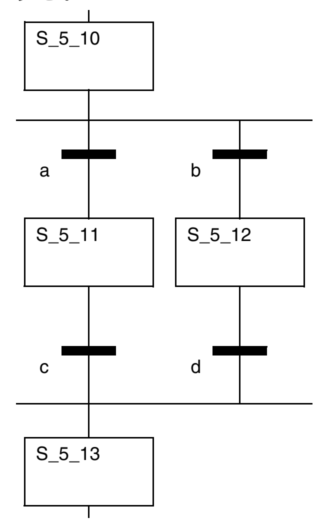
选择串

用户可以在多令牌的选择分支中为转换条件进行行为定义。
可以进行以下操作：

- 从左向右进行处理，在第一个有效转换 (1- 关闭 -n- 选择) 以后停止。这相当于单令牌中的选择串行为 (参见*选择串， 356 页*)。
- 对所有选择分支的转换进行并行处理 (x- 关闭 -n- 选择)

x- 关闭 -n- 选择：

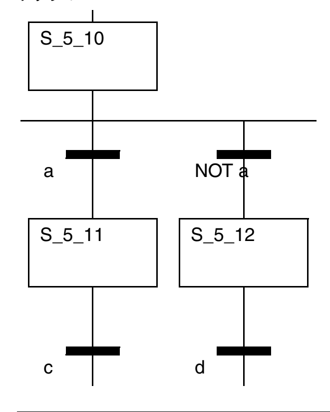
在多令牌模式下，可以从转换进行多个并行切换操作 (x- 关闭 -n- 选择)。要运行的分支是由跟在选择分支后面的转换条件结果决定的。分支的所有转换都会被处理。满足条件转换的所在分支会被激活。如果没有任何转换，当前的步会保持设置状态。



<table>
<tr><th colspan="2">如果 ...</th><th colspan="2">那么</th></tr>
<tr><td colspan="2">如果 S_5_10 有效，并且转换条件 a 为真，b 为假，</td><td colspan="2">那么会有一个顺序从 S_5_10 运行到 S_5_11。</td></tr>
<tr><td colspan="2">如果 S_5_10 有效，并且转换条件 a 为假，b 为真，</td><td colspan="2">那么会有一个顺序从 S_5_10 运行到 S_5_12。</td></tr>
<tr><td colspan="2">如果 S_5_10 有效，并且转换条件 a 和 b 都为真，</td><td colspan="2">那么会有一个顺序从 S_5_10 运行到 S_5_11 和 S_5_12。</td></tr>
<tr><td colspan="4">同时激活两个选择分支会创建第二个令牌。这两个令牌现在以并行的方式运行，也就是说，S_5_11 和 S_5_12 同时有效。</td></tr>
<tr><th colspan="2">令牌 1 (S_5_11)</th><th colspan="2">令牌 2 (S_5_12)</th></tr>
<tr><th>如果 ...</th><th>那么</th><th>如果 ...</th><th>那么</th></tr>
<tr><td>如果转换条件 c 为真，</td><td>那么会有一个顺序从 S_5_11 运行到 S_5_13。</td><td>如果转换条件 d 为真，</td><td>那么会有一个顺序从 S_5_12 运行到 S_5_13。</td></tr>
<tr><td colspan="4">如果因为转换条件 c 被激活而导致 S_5_13 仍然有效 (令牌 1)，那么令牌 2 就会被中止，该串会以单令牌的形式被进一步处理。如果 If S_5_13 不再有效 (令牌 1)，那么它就被令牌 2 重新激活，两个令牌以并行方式同时运行 (多令牌)。</td></tr>
</table>

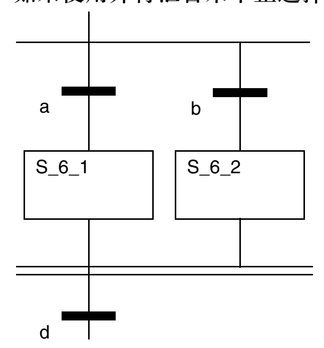
如果在这种操作模式下，只想切换选择分支，那么必须通过转换逻辑进行明确的定义。

例子：



通过并行汇合中止选择分支

如果使用并行汇合来中止选择分支，可能会把串封闭。
如果使用并行汇合来中止选择分支：



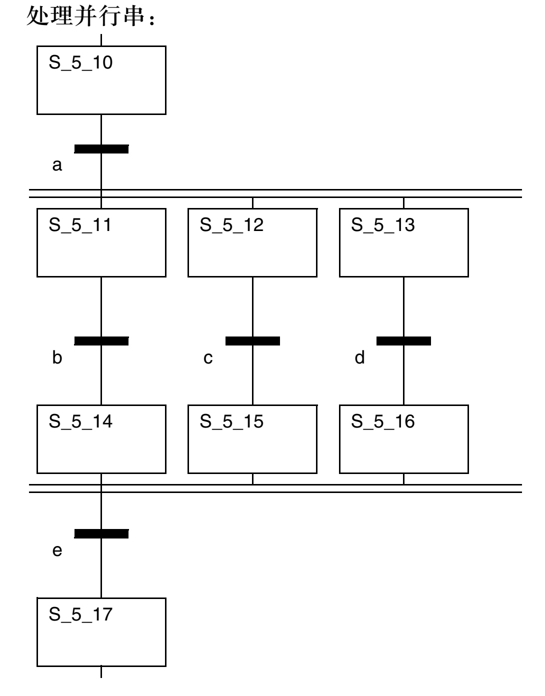
<table>
<tr><th>如果 ...</th><th>那么</th></tr>
<tr><td>如果转换条件 a 为真，b 为假，</td><td>那么会有一个顺序运行到 S_6_1。</td></tr>
<tr><td colspan="2">考虑到 S_6_1 和 S_6_2 与一个并行汇合相链接，该分支不能离开，因为 S_6_1 和 S_6_2 无论如何也不能同时有效。
(或者 S_6_1 通过转换条件 a 激活，或者 S_6_2 通过转换条件 b 激活。)
所以 S_6_1 和 S_6_2 也不能同时有效。串被封闭了。
这个功能块可以重新移动，比如使用第二个令牌，运行转换 b。</td></tr>
</table>

并行串

并行串

通过并行分支，对一个单一的转换操作，会同时触发多个 (最多 32 个) 步 (分支)。执行的顺序是从左到右。在这次整体触发以后，每一个分支会独立处理。这对单令牌和多令牌同样适用。

处理并行串：

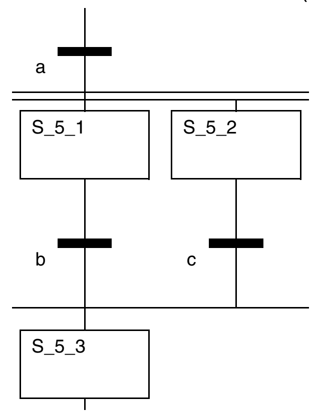


如果 ...	那么
如果 S_5_10 有效，并且隶属于公共转换的转换条件 a 也为真，	那么会有一个顺序从 S_5_10 运行到 S_5_11， S_5_12 和 S_5_13。
如果步 S_5_11， S_5_12 和 S_5_13 被激活，	那么各个串之间就会彼此独立运行。
如果 S_5_14， S_5_15 和 S_5_16 同时有效，并且隶属于公共转换的转换条件 e 也为真，	那么会有一个顺序从 S_5_14， S_5_15 和 S_5_16 运行到 S_5_17。

通过选择汇合中止并行分支

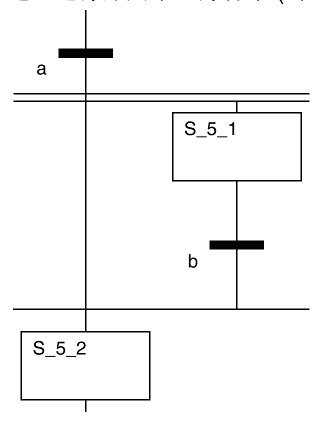
在多令牌模式下，除了并行汇合，还可以使用选择汇合来中止并行分支。

通过选择汇合来中止并行串 (第 1 种变化)：



<table>
<tr><th colspan="2">如果 ...</th><th colspan="2">那么</th></tr>
<tr><td colspan="2">如果转换条件 a 为真，</td><td colspan="2">那么会有一个顺序从 a 到 S_5_1 和 S_5_2 的运行。</td></tr>
<tr><td colspan="2">如果步 S_5_1 和 S_5_2 被激活，</td><td colspan="2">那么各个串之间就会彼此独立运行。</td></tr>
<tr><td colspan="2">如果转换条件 b 为真，并且 c 为假，</td><td colspan="2">那么会有一个顺序运行到 S_5_3。</td></tr>
<tr><td colspan="4">在并行串以外的选择汇合上运行的顺序创建了第二个令牌。两个令牌以并行的方式运行，也就是说， S_5_2 和 S_5_3 同时有效。</td></tr>
<tr><td colspan="2">令牌 1 (S_5_3)</td><td colspan="2">令牌 2 (S_5_2)</td></tr>
<tr><td>如果 ...</td><td>那么</td><td>如果 ...</td><td>那么</td></tr>
<tr><td colspan="2">步 S_5_3 有效。</td><td colspan="2">步 S_5_2 有效。</td></tr>
<tr><td colspan="2"></td><td>如果转换条件 c 为真，</td><td>那么会有一个顺序运行到 S_5_3。</td></tr>
<tr><td colspan="4">如果 S_5_3 仍然有效 (令牌 1)，那么第 2 个令牌会被中止，该串会以单令牌形式被继续处理。
如果 S_5_3 不再有效 (令牌 1)，那么它会被令牌 2 重新激活，两个令牌会以并行方式继续运行 (多令牌)。</td></tr>
</table>

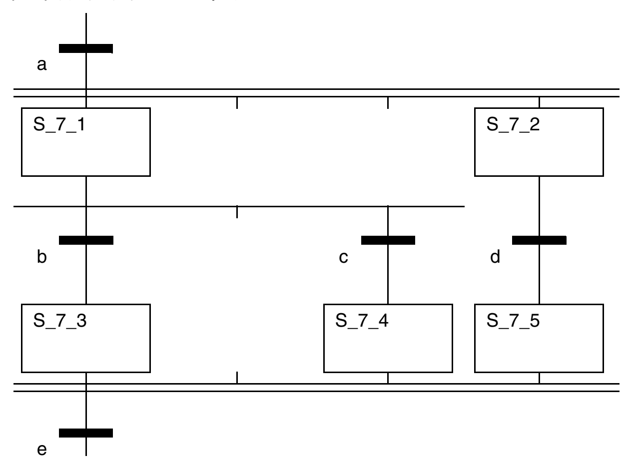
通过选择分支中止并行串 (第 2 种变化)：



<table>
<tr><th colspan="2">如果 ...</th><th colspan="2">那么</th></tr>
<tr><td colspan="2">如果转换条件 a 为真，</td><td colspan="2">那么会有一个顺序运行到 S_5_1 和 S_5_2。</td></tr>
<tr><td colspan="4">在并行串以外的选择汇合上运行的顺序创建了第二个令牌。两个令牌以并行的方式运行，也就是说， S_5_1 和 S_5_2 同时有效。</td></tr>
<tr><td colspan="2">令牌 1 (S_5_2)</td><td colspan="2">令牌 2 (S_5_1)</td></tr>
<tr><td>如果 ...</td><td>那么</td><td>如果 ...</td><td>那么</td></tr>
<tr><td colspan="2">步 S_5_2 有效。</td><td colspan="2">步 S_5_1 有效。</td></tr>
<tr><td colspan="2"></td><td>如果转换条件 b 为真，</td><td>那么会有一个顺序运行到 S_5_2。</td></tr>
<tr><td colspan="4">如果 S_5_2 仍然有效 (令牌 1)，那么第 2 个令牌会被中止，该串会以单令牌形式被继续处理。
如果 S_5_2 不再有效 (令牌 1)，那么它会被令牌 2 重新激活，两个令牌会以并行方式继续运行 (多令牌)。</td></tr>
</table>

在并行串中使用选择分支

如果在一个并行串中使用了一个单一的选择分支，它可能会使串封闭。
在并行串中使用选择分支：



<table>
<tr><th>如果 ...</th><th>那么</th></tr>
<tr><td>如果转换条件 a 为真，</td><td>那么会有一个顺序运行到 S_7_1 和 S_7_2。</td></tr>
<tr><td>如果步 S_7_1 和 S_7_2 被激活，</td><td>那么各个串之间就会彼此独立运行。</td></tr>
<tr><td>如果转换条件 d 为真，</td><td>那么会有一个顺序运行到 S_7_5。</td></tr>
<tr><td>如果转换条件 b 为真，</td><td>那么会有一个顺序运行到 S_7_3。</td></tr>
<tr><td colspan="2">考虑到 S_7_3， S_7_4 和 S_7_5 与一个并行汇合相链接，该并行串不能离开，因为 S_7_3 和 S_7_4 无论如何也不能同时有效。(或者 S_7_3 通过转换条件 b 激活，或者 S_7_4 通过转换条件 c 激活。)
所以 S_7_3， S_7_4 和 S_7_5 也不能同时有效。串被封闭了。
这个功能块可以再次移动，比如第二个令牌使转换 c 运行起来。</td></tr>
</table>

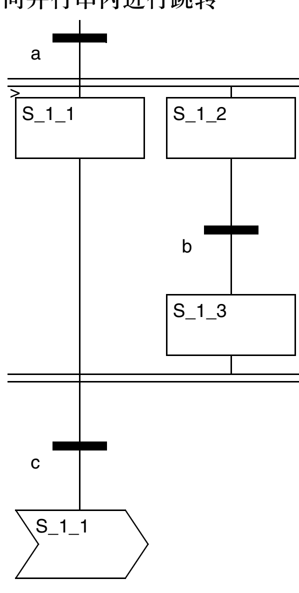
跳转到并行串中

描述

在多令牌模式下，可以使用向并行串中或者从并行串向外的跳转功能。
向并行串中的跳转不会激活所有分支。因为对于并行汇合后面的转换来说，只有当直接位于它前方的所有步都激活了以后，它才会被评估，所以永远也无法离开并行串，串被封闭了。

向并行串内进行跳转

向并行串内进行跳转



如果 ...	那么
如果转换条件 a 为真，	那么会有一个顺序运行到 S_1_1 和 S_1_2。
如果步 S_1_1 和 S_1_2 被激活，	那么各个串之间就会彼此独立运行。
如果 S_1_2 有效，并且转换条件 b 为真，	那么会有一个顺序从 S_1_2 运行到 S_1_3。
如果 S_1_1 和 S_1_3 有效，并且隶属于公共转换的转换条件 c 为真，	那么会有一个顺序从 S_1_1 和 S_1_3 运行到到 S_1_1 的跳转。
如果 S_1_1 被跳转所激活，	那么只有来自 S_1_1 的分支有效。来自 S_1_2 的分支无效。
因为 S_1_1 和 S_1_3 不会同时有效，串不能继续运行，它被封闭了。这个功能块可以重新移动，比如通过第二个令牌，置位来重新激活 S_1_2 步。	

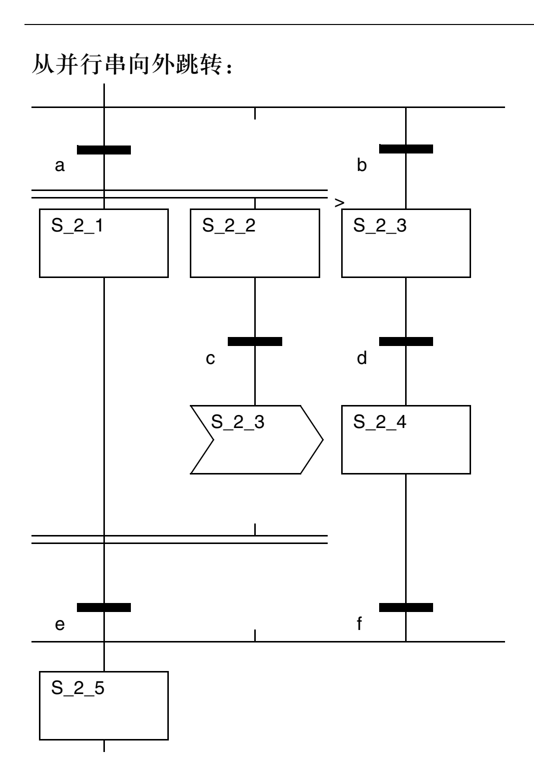
从并行串向外跳转

介绍

在多令牌模式下，可以使用向并行串中或者从并行串向外的跳转功能。
在各种情况下，都会产生附加的令牌。

从并行串向外跳转

从并行串向外跳转：

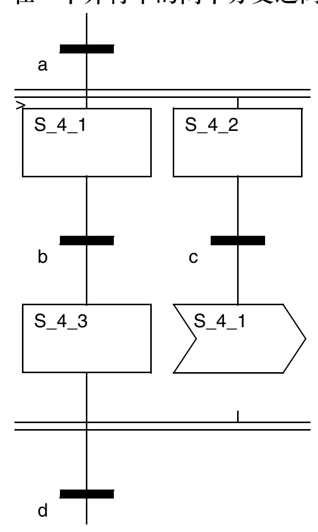


如果 ...	那么
如果转换条件 a 为真，并且 b 为假，	那么会有一个顺序运行到 S_2_1 和 S_2_2。
如果步 S_2_1 和 S_2_2 被激活，	那么各个串之间就会彼此独立运行。
如果转换条件 c 为真，	那么就会有一个到 S_2_3 的跳转。
在并行串外面的跳转会产生第二个令牌。两个令牌会以并行的方式运行，也就是说， S_2_1 和 S_2_3 同时有效。	

令牌 1 (S_2_1)		令牌 2 (S_2_3)	
如果 ...	**那么**	**如果 ...**	**那么**
如果转换条件 e 为真，	那么会有一个顺序运行到 S_2_5。	如果转换条件 d 为真，	那么会有一个顺序运行到 S_2_4。
		如果转换条件 f 为真，	那么会有一个顺序运行到 S_2_5。
如果因为转换条件 e 被激活， S_2_5 仍然有效 (令牌 1)，那么第 2 个令牌会被中止，该串会以单令牌形式被继续处理。 如果 S_2_5 不再有效 (令牌 1)，那么它会被令牌 2 重新激活，两个令牌会以并行方式继续运行 (多令牌)。			

在一个并行串的两个分支之间进行跳转

在一个并行串的两个分支之间进行跳转：

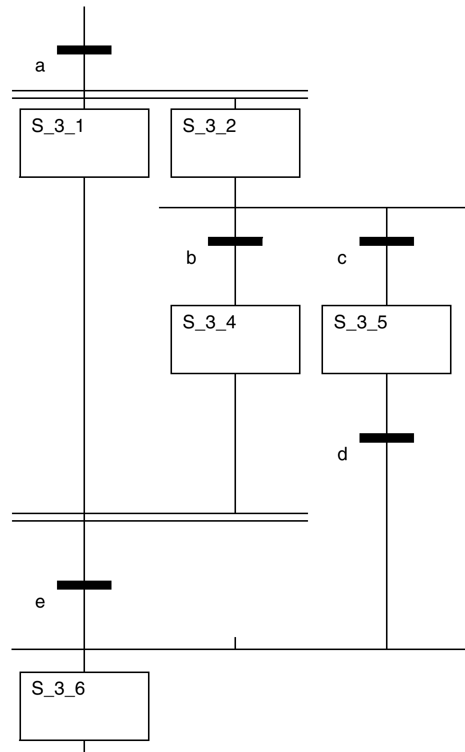


如果 ...	那么
如果转换条件 a 为真，	那么会有一个顺序运行到 S_4_1 和 S_4_2。
如果步 S_4_1 和 S_4_2 被激活，	那么各个串之间就会彼此独立运行。
如果转换条件 b 为真，	那么会有一个顺序运行到 S_4_3。
如果转换条件 c 为真，	那么就会有一个到 S_4_1 的跳转
在分支串外面的跳转会产生第二个令牌。两个令牌会以并行的方式运行，也就是说， S_4_3 和 S_4_1 同时有效。	

令牌 1 (S_4_3)		**令牌 2 (S_4_1)**	
如果 ...	**那么**	**如果 ...**	**那么**
步 S_4_3 会被处理		步 S_4_1 会被处理	
		如果转换条件 b 为真，	那么会有一个顺序运行到 S_4_3。
如果在被令牌 2 激活的期间内，步 S_4_3 仍然有效 (令牌 1)，那么第 2 个令牌会被中止，该串会以单令牌形式被继续处理。 如果 S_4_3 不再有效 (令牌 1)，那么它会被令牌 2 重新激活，两个令牌会以并行方式继续运行 (多令牌)。 在这两种情况下，如果转换条件 d 为真，就会离开并行串。			

通过选择分支离开并行串

通过选择分支离开并行串：



<table>
<tr><th colspan="2">如果 ...</th><th colspan="2">那么</th></tr>
<tr><td colspan="2">如果转换条件 a 为真，</td><td colspan="2">那么会有一个顺序运行到 S_3_1 和 S_3_2。</td></tr>
<tr><td colspan="2">如果步 S_3_1 和 S_3_2 被激活，</td><td colspan="2">那么各个串之间就会彼此独立运行。</td></tr>
<tr><td colspan="2">如果转换条件 b 为假，并且 c 为真，</td><td colspan="2">那么会有一个顺序运行到 S_3_5。</td></tr>
<tr><td colspan="4">在分支串外面的选择分支上运行的顺序会产生第二个令牌。两个令牌会以并行的方式运行，也就是说， S_3_1 和 S_3_5 同时有效。</td></tr>
<tr><td colspan="2">令牌 1 (S_3_1)</td><td colspan="2">令牌 2 (S_3_5)</td></tr>
<tr><td>如果 ...</td><td>那么</td><td>如果 ...</td><td>那么</td></tr>
<tr><td colspan="2">因为 S_3_4 不能变为有效，所以 S_3_1 会保持 (令牌 1) 有效。</td><td>如果转换条件 d 为真，</td><td>那么会有一个顺序运行到 S_3_6。</td></tr>
<tr><td colspan="4">如果转换条件 a 为真，那么会有一个顺序运行到 S_3_1 和 S_3_2。那么第 2 个令牌会被中止，该串会以单令牌形式被继续处理。</td></tr>
<tr><td colspan="4">如果转换条件 a 为真，
那么会有一个顺序运行到 S_3_1 和 S_3_2。</td></tr>
<tr><td></td><td></td><td>如果转换条件 b 为真，并且 c 为假，</td><td>那么会有一个顺序运行到 S_3_4。</td></tr>
<tr><td colspan="4">因为 S_3_4 不能变为有效，所以 S_3_1 保持 (令牌 1) 有效，直到在 S_3_2 (令牌 2) 上出现了一个顺序，转换 b 为真。
两个令牌会以并行方式继续运行 (多令牌)。</td></tr>
</table>

指令表 IL

14

介绍

概述

本章描述了符合 IEC 61131 的编程语言指令表 IL。

本章内容

本章包含以下各节：

14.1 关于 IL 指令表的常规信息

介绍

概述

本节给出了 IL 指令表的综述

本节内容

本节包含以下主题：

关于 IL 指令表的常规信息

介绍

通过指令表编程语言 (IL)，您可以以有条件或者无条件的方式来调用功能块，进行赋值，在一个代码段中以有条件或者无条件的方式进行跳转。

指令

一个指令表包含一系列指令。每一个指令都要从新的一行开始，并包含以下内容：

- 一个运算符 (参见*操作数， 384 页*)，
- 如果需要的话，一个限定词 (参见*限定词， 386 页*) 以及
- 如果需要的话，一个或多个操作数 (参见*操作数， 384 页*)。

如果要使用多个操作数，它们之间应该用逗号分开。在指令前可以使用一个标号 (参见*标记和跳转， 398 页*)。这个标号后面会带有一个冒号。指令的后面可以带有一个注释 (参见*注释， 400 页*) 。

例子

标记　　运算符　　操作数

```
START:  LD      A    (* Key 1 *)
        ANDN    B    (* and not key 2 *)
        ST      C    (* Ventilator on *)
```

限定词　　注释

编程语言的结构

IL 被称为面向累加器的语言，也就是说，它的每一个指令都会用到或者更改累加器 (内部高速缓存的一种形式) 的内容。 IEC 61131 将这种累加器称为 “结果”。
出于这个原因，一个指令表应该总是以 LD 运算符开始 (“加载累加器命令”)。
加法的例子：

指令	描述
LD 10	在累加器中加载 10。
ADD 25	把 “25”加到数字存储器的内容中去。
ST A	结果被存储在变量 A 中。 变量 A 和累加器中的内容现在是 35。后面的指令如果不以 LD 开头，那么那就会将 “35”作为累加器的内容。

同样，比较操作也会经常用到累加器。比较的布尔结果被存储在累加器中，作为累加器的当前内容。
比较的例子：

指令	描述
LD B	在累加器中加载数值 B。
GT 10	将 10 与累加器中的内容相比。
ST A	比较的结果存储在变量 A 中。 如果 B 小于或等于 10，变量 A 的数值以及累加器的内容都是 0 (FALSE)。如果 B 大于 10，变量 A 的数值以及累加器的内容都是 1 (TRUE)。

代码段大小

一个指令行的长度不能超过 300 个字符。
在编程环境中没有限制 IL 代码段的长度。 IL 代码段的长度只受 PLC 内存的限制。

语法

标识符和关键字不区分大小写。
空格键和 tab 键对语法没有影响，可以随便使用，
例外：不允许使用 - 空格和 tab 键的情况

- 关键字，
- 字面值，
- 数值，
- 标识符，
- 变量，以及
- 限制符组合 [比如 (* 用于注释)]。

执行顺序

指令是自上而下逐行执行的。可以通过括号来改变这个顺序。

如果 A， B，C 和 D 的数值是 1，2，3 和 4，并按如下方式运算：

```
LD A
ADD B
SUB C
MUL C
ST E
```

那么在 E 中给出的结果就是 0。

如果按照以下方式运算：

```
LD A
ADD B
SUB(
LD C
MUL D
)
ST E
```

那么在 E 中给出的结果就是 -9。

错误行为

在执行一个表达式的时候，如下情况被认定为出错：

- 尝试除 0 操作。
- 操作数中不含有操作所需的正确数据类型。
- 数字运算的结果超过了该数据类型的取值范围。

遵循的 IEC 标准

关于 IL 编程语言所遵循的 IEC 标准，请参见 “遵循的 IEC 标准”。

操作数

介绍

运算符用来对操作数进行操作。
操作数可以是：
- 一个地址
- 一个字面值
- 一个变量
- 一个多元素变量
- 一个多元素变量的元素
- 一个 EFB/DFB 输出 或者
- 一个 EFB/DFB 调用。

数据类型

操作数和累加器当前的内容必须是同一类型。如果要处理不同类型的操作数，必须预先进行类型转换。
在下面的例子中，整数变量 i1 被转换为一个实数变量，然后才与实数变量 r4 相加。

```
LD  i1
INT_TO_REAL
ADD r4
ST  r3
```

这个规则有一个例外，那就是数据类型 TIME 的变量可以和数据类型 INT， DINT， UINT 或者 UDINT 的变量进行乘除运算。
允许的运算：
- ```
  LD  timeVar1
  DIV dintVar1
  ST  timeVar2
  ```
- ```
  LD  timeVar1
  MUL intVar1
  ST  timeVar2
  ```
- ```
  LD  timeVar1
  MUL 10
  ST  timeVar2
  ```

该功能被 IEC 61131-3 列为 “不推荐使用”的内容。
  ```

直接使用地址

地址可以直接使用（事先不需要给出声明）。在这种情况下，数据类型会直接分配给地址。分配过程是通过 “大前缀” 来完成的。

下面的表格给出了各种大前缀：

大前缀 / 符号	例子	数据类型
没有前缀	%I10, %CH203.MOD, %CH203.MOD.ERR	BOOL
X	%M**X**20	BOOL
B	%Q**B**102.3	BYTE
W	%K**W**43	INT
D	%Q**D**100	DINT
F	%M**F**100	REAL

使用其他数据类型

如果要把其他数据类型分配为一个地址的缺省数据类型，那么必须给出明确的声明。这个变量声明可以使用变量编辑器来轻松地实现。地址的数据类型不能直接在 ST 代码段中进行声明（比如说，声明 AT %MW1：UINT; 是不允许的）。下面的变量在变量编辑器中进行了声明：

```
UnlocV1: ARRAY [1..10] OF INT;
LocV1:   ARRAY [1..10] OF INT AT %MW100;
LocV2:   TIME AT %MW100;
```

这样一来，下面的调用就拥有了正确的语法：

```
%MW200 := 5;
LD LocV1[%MW200]
ST UnlocV1[2]

LD t#3s
ST LocV2
```

访问字段变量

在访问数组变量 (ARRAY) 的时候，在索引项中只能使用 INT， DINT， UINT 和 UDINT 类型的变量和字面值。

如果一个 ARRAY 元素的下限是负值，那么它的索引就可以是负值。

例子：保存一个变量

```
LD var1[i]
ST var2.otto[4]
```

限定词

介绍

限定词用来影响运算符的执行 (参见*运算符， 388 页*)。

限定词列表

限定词列表：

限定词	使用运算符的数据类型	描述
N	BOOL, BYTE, WORD, DWORD	限定词 N 用来把操作数的数值逐位反转。 **例子：**在这个例子中，如果 A 为 1 并且 B 为 0，那么 C 就是 1。 LD A AND**N** B ST C
C	BOOL	如果限定词累加器的值是 1 (TRUE)，那么 C 用来执行相关指令。 **例子：**在这个例子中，只有当 A 为 1 (TRUE)，并且 B 也为 1 (TRUE) 的情况下，在 START 后面的跳转才会被执行。 LD A AND B JMP**C** START
CN	BOOL	如果限定词 C 和 N 不在一起，那么只有当累加器的数值为布尔 0 (FALSE) 时，才会执行相关的操作。 **例子：**在这个例子中，只有当 A 为 0 (FALSE)，并且 B 也为 0 (TRUE) 的情况下，在 START 后面的跳转才会被执行。 LD A AND B JMP**CN** START

限定词	使用运算符的数据类型	描述
(	所有	左括号限定词 (用来回溯操作数的运算，直到右括号运算符为止)。右括号运算的数量必须和左括号限定词的数量一致。括号可以嵌套。 **例子：**在这个例子中，如果 C 与 / 或 D 是 1，并且 A 与 B 是 1，那么 E 就为 1。 LD A AND B AND**(** C OR D) ST E 这些例子也可以用以下方式来编写： LD A AND B AND**(** LD C OR D) ST E

运算符

介绍

运算符是一个符号，它用来：
- 执行算术运算
- 执行一个配置操作或者
- 用一个基本功能块 - DFB 或者子程序。

运算符是通用的，也就是说，它们会与操作数的数据类型自动匹配。

加载和保存运算符

IL 编程语言加载和保存运算符：

运算符	限定词	描述	操作数	描述
LD	N (仅用于 BOOL, BYTE, WORD 或者 DWORD 数据类型的操作数)	把操作数的数值加载到数字存储器中	任何数据类型的字面值，变量，直接地址	一个运算的数值会通过 LD 加载到累加器中。累加器中数据的大小会根据运算的数据类型进行自动调整。对于导出数据类型，情况也是一样的。 **例子：**在这个例子中， A 的数值被加载到累加器中，与 B 值相加，并将结果保存在 E 中。 **LD** A ADD B ST E

运算符	限定词	描述	操作数	描述
ST	N (仅用于 BOOL, BYTE, WORD 或者 DWORD 数据类型的操作数)	把累加器的数值加载到操作数中	任何数据类型的字面值，变量，直接地址	累加器的当前值通过 ST 指令存储在操作数中。操作数的数据类型必须与累加器的 “数据类型” 一致。 **例子：**在这个例子中， A 的数值被加载到累加器中，与 B 值相加，并将结果保存在 E 中。 LD A ADD B **ST** E 如果 ST 后面带有 LD 指令， “原来的” 结果会进行进一步的运算处理。 **例子：**在这个例子中， A 的数值被加载到累加器中，与 B 值相加，并将结果保存在 E 中。 E 的数值 (累加器当前的内容) 减去 3，其结果保存在 C 中。 LD A ADD B **ST** E SUB 3 **ST** C

置位和复位运算符　置位和复位 IL 编程语言的运算符：

运算符	限定词	描述	操作数	描述
S	-	当累加器的内容为 1 时，将操作数设为 1。	BOOL 数据类型的变量，直接地址。	当数字存储器的当前内容为布尔 1 时， S 把操作数设为 “1”。 **例子：**在这个例子中， A 的数值被加载到累加器中，如果累加器的内容 (A 的数值) 为 1，那么 OUT 会被设定为 1。 LD A **S** OUT 通常情况下这个运算符与复位运算符共同使用。 **例子：**这个例子给出了一个 RS 触发器 (复位优先)，通过两个布尔变量 A 和 C 进行控制。 LD A **S** OUT LD C R OUT

运算符	限定词	描述	操作数	描述
R	-	当累加器的内容为 1 时，将操作数设为 0	BOOL 数据类型的变量，直接地址。	当累加器的当前内容为布尔 1 时， R 把操作数设为 “0”。 **例子：**在这个例子中， A 的数值被加载到累加器中，如果累加器的内容 (A 的数值) 为 1，那么 OUT 会被置为 0。 LD A **R** OUT 通常情况下这个运算符与置位运算符共同使用。 例子：这个例子给出了一个 RS 触发器 (置位优先)，通过两个布尔变量 A 和 C 进行控制。 LD A **R** OUT LD C S OUT

逻辑运算符

IL 编程语言逻辑运算符：

运算符	限定词	描述	操作数	描述
AND	N, N(, (	逻辑与	BOOL， BYTE， WORD 或者 DWORD 数据类型的字面值，变量，直接地址	通过 AND 运算符，可以在累加器的内容和操作数之间建立一个逻辑与链接。 这个链接通过 BYTE， WORD 和 DWORD 数据类型逐位建立。 **例子：**在这个例子中，如果 A， B 和 C 是 1， D 就是 1。 LD A **AND** B **AND** C ST D
OR	N, N(, (	逻辑或	BOOL， BYTE， WORD 或者 DWORD 数据类型的字面值，变量，直接地址	通过 OR 运算符，可以在累加器的内容和操作数之间建立一个逻辑或链接。 这个链接通过 BYTE， WORD 和 DWORD 数据类型逐位建立。 **例子：**在这个例子中，如果 A 或 B 是 1，并且 C 是 1，那么 D 就是 1。 LD A **OR** B **OR** C ST D

运算符	限定词	描述	操作数	描述
XOR	N, N(, (	逻辑异或	BOOL， BYTE，WORD 或者 DWORD 数据类型的字面值，变量，直接地址	通过 XOR 运算符，可以在累加器的内容和操作数之间建立一个逻辑异或链接。 如果链接的操作数超过两个，那么如果 1 状态的数量是奇数，结果就是 1，如果 1 状态的数量是偶数，结果就是 0。 这个链接通过 BYTE， WORD 和 DWORD 数据类型逐位建立。 **例子：**在这个例子中，如果 A 和 B 具有同样的状态 (都是 0 或 1)，那么 D 就是 0。 LD A **XOR** B ST D 如果链接的操作数超过两个，那么如果 1 状态的数量是奇数，结果就是 1，如果 1 状态的数量是偶数，结果就是 0。 **例子：**在这个例子中，如果 1 个或 3 个操作数为 1，那么 F 就是 1，如果 0 个， 2 个或者 4 个操作数为 1，那么 F 就是 0。 LD A **XOR** B **XOR** C **XOR** D **XOR** E ST F
NOT	-	逻辑否	BOOL， BYTE，WORD 或者 DWORD 数据类型的数字存储器数字存储器内容。	数字存储器内容通过 NOT 求反。 **例子：**在这个例子中，如果 A 是 0，那么 B 就是 1，如果 A 是 1，那么 B 就是 0。 LD A **NOT**, ST B

算术运算符

IL 编程语言算术运算符：

运算符	限定词	描述	操作数	描述
ADD	(	加法	INT， DINT， UINT， UDINT， REAL或者TIME数据类型的字面值，变量，直接地址。	通过 ADD 运算符，操作数的值与累加器的内容相加。 **例子：**这个例子按照 D = A + B + C 进行运算 LD A **ADD** B **ADD** C ST D
SUB	(	减法	INT， DINT， UINT， UDINT， REAL或者TIME数据类型的字面值，变量，直接地址 E。	通过 SUB 运算符，累加器的内容会减去操作数的值。 **例子：**这个例子按照 D = A - B - C 进行运算 LD A **SUB** B **SUB** C ST D
MUL	(	乘法	INT， DINT， UINT， UDINT， REAL或者TIME数据类型的字面值，变量，直接地址。	通过 MUL 运算符，累加器的内容会与操作数的值相乘。 **例子：**这个例子按照 D = A * B * C 进行运算 LD A **MUL** B **MUL** C ST D **Note:** The MULTIME function in the obsolete library is available for multiplications involving the data type Time.
DIV	(	除法	INT， DINT， UINT， UDINT， REAL或者TIME数据类型的字面值，变量，直接地址。	通过 DIV 运算符，累加器的内容会除以操作数的值。 **例子：**这个例子按照 D = A / B / C 进行运算 LD A **DIV** B **DIV** C ST D **Note:** The DIVTIME function in the obsolete library is available for divisions involving the data type Time.

运算符	限定词	描述	操作数	描述
MOD	(	模除	INT， DINT，UINT， UDINT，REAL 或者 TIME 数据类型的字面值，变量，直接地址。	在 MOD 运算中，第一个操作数的值会除以第二个操作数的值，所得的余数 (模数) 作为返回的结果。 **例子：**在这个例子中 ● 如果 A 是 7， B 是 2，那么 C 是 1 ● 如果 A 是 7， B 是 -2，那么 C 是 1 ● 如果 A 是 -7， B 是 2，那么 C 是 -1 ● 如果 A 是 -7， B 是 -2，那么 C 是 -1 LD A **MOD** B ST C

比较运算符

IL 编程语言的比较运算符：

运算符	限定词	描述	操作数	描述
GT	(	比较：>	BOOL， BYTE，WORD，DWORD，STRING，INT，DINT， UINT，UDINT， REAL，TIME，DATE，DT 或者 TOD 数据类型的字面值，变量，直接地址 。	通过 GT 运算符，累加器内容会与操作数的内容进行比较。如果累加器的内容大于操作数的内容，那么结果就是一个布尔 1。如果累加器内容小于 / 等于操作数的内容，那么结果就是一个布尔 0。 **例子：**在这个例子中，如果 A 大于 10， D 的值就是 1，否则 D 的值就是 0。 LD A **GT** 10 ST D
GE	(	比较：>=	BOOL， BYTE，WORD，DWORD，STRING，INT，DINT，UINT，UDINT， REAL，TIME，DATE，DT 或者 TOD 数据类型的字面值，变量，直接地址。	通过 GE 运算符，累加器内容会与操作数的内容进行比较。如果累加器的内容大于 / 等于操作数的内容，那么结果就是一个布尔 1。如果累加器的内容小于操作数的内容，那么结果就是一个布尔 0。 **例子：**在这个例子中，如果 A 大于或等于 10， D 的值就是 1，否则 D 的值就是 0。 LD A **GE** 10 ST D

运算符	限定词	描述	操作数	描述
EQ	(	比较：=	BOOL， BYTE，WORD，DWORD，STRING， INT，DINT，UINT，UDINT， REAL，TIME，DATE，DT 或者 TOD 数据类型的字面值，变量，直接地址。	通过 EQ 运算符，累加器内容会与操作数的内容进行比较。如果累加器的内容等于操作数的内容，那么结果就是一个布尔 1。如果累加器的内容不等于操作数的内容，那么结果就是一个布尔 0。 **例子：**在这个例子中，如果 A 等于 10， D 的值就是 1，否则 D 的值就是 0。 LD A **EQ** 10 ST D
NE	(	比较：<>	BOOL， BYTE，WORD，DWORD，STRING， INT，DINT，UINT，UDINT， REAL，TIME，DATE，DT 或者 TOD 数据类型的字面值，变量，直接地址。	通过 NE 运算符，累加器内容会与操作数的内容进行比较。如果累加器的内容不等于操作数的内容，那么结果就是一个布尔 1。如果累加器的内容等于操作数的内容，那么结果就是一个布尔 0。 **例子：**在这个例子中，如果 A 不等于 10， D 的值就是 1，否则 D 的值就是 0。 LD A **NE** 10 ST D
LE	(	比较：<=	BOOL， BYTE，WORD， DWORD，STRING， INT，DINT，UINT，UDINT， REAL，TIME，DATE，DT 或者 TOD 数据类型的字面值，变量，直接地址。	通过 LE 运算符，累加器内容会与操作数的内容进行比较。如果累加器的内容小于 / 等于操作数的内容，那么结果就是一个布尔 1。如果累加器的内容大于操作数的内容，那么结果就是一个布尔 0。 **例子：**在这个例子中，如果 A 小于或等于 10， D 的值就是 1，否则 D 的值就是 0。 LD A **LE** 10 ST D

运算符	限定词	描述	操作数	描述
LT	(	比较：<	BOOL， BYTE， WORD，DWORD， STRING，INT， DINT，UINT， UDINT， REAL， TIME，DATE，DT 或者 TOD 数据类型的字面值，变量，直接地址。	通过 LT 运算符，数字存储器内容会与操作数的内容进行比较。如果数字存储器的内容小于操作数的内容，那么结果就是一个布尔 1。如果数字存储器的内容大于 / 等于操作数的内容，那么结果就是一个布尔 0。 **例子：**在这个例子中，如果 A 小于 10， D 的值就是 1，否则 D 的值就是 0。 LD A **LT** 10 ST D

调用运算符

IL 编程语言的调用运算符：

运算符	限定词	描述	操作数	描述
CAL	C，CN (只有当累加器的内容是 BOOL 数据类型时才能使用)	调用一个功能块， DFB 或者子程序	功能块， DFB 或者子程序的实例名称	CAL 用来以有条件的方式或者无条件的方式调用一个功能块， DFB 或者子程序。 请同时参见*调用基本功能块和导出功能块， 407 页以及子程序调用， 397 页*
FUNCTIONNAME	-	执行一个功能	字面值，变量，直接地址 (数据类型取决于功能)	通过一个功能名称来执行一个功能。请同时参见*基本功能调用， 402 页*
PROCEDURENAME	-	执行一个功能程序	字面值，变量，直接地址 (数据类型取决于功能程序)	通过一个功能程序名称来执行一个功能程序。 请同时参见*调用功能程序， 416 页*

结构运算符

IL 编程语言的结构运算符：

运算符	限定词	描述	操作数	描述
JMP	C，CN（只有当累加器的内容是 BOOL 数据类型时才能使用）	跳转到标号	TAG	通过 JMP 运算符，可以实现到一个标号有条件或者无条件的跳转。 请同时参见*标号和跳转， p.398*
RET	C， CN（只有当累加器的内容是 BOOL 数据类型时才能使用）	返回到下一个最高程序结构单元	-	每一个子程序和 DFB（导出功能块）在处理完毕后都会退出，也就是说，返回到调用的主程序。如果要提早离开子程序 /DFB，可以通过 RET （返回）强制返回主程序。 RET 只能用在子程序或者 DFB 中。它们不能用在主程序中。
)	-	处理延后的运算	-	通过右括号 ），可以开始处理延后的运算符。右括号运算符的数量必须与左括号限定词的数量一致。括号可以嵌套。 **例子：**在这个例子中，如果 C 和 / 或 D 是 1，并且 A 和 B 是 1，那么 E 就是 1。 LD A AND B AND**(** C OR D **)** ST E

子程序调用

调用子程序

子程序调用中包含 `CAL` 运算符，它的后面是子程序代码段的名称，名称后面则是一个空的参数列表 (可选)。

子程序调用没有返回值。

要调用的子程序必须和调用的 IL 代码段位于同一个任务内。子程序也可以从子程序内部进行调用。

比如

```
ST A
CAL SubroutineName ()
LD B
```

或

```
ST A
CAL SubroutineName
LD B
```

子程序是 IEC 61131-3 的补充内容，必须被明确激活。

在 SFC 动作代码段中，只有在激活了多令牌操作的情况下，才能使用子程序调用。

标号和跳转

介绍

标号是跳转的目标。

标号的属性：

标号的属性：
- 标号必须是一行中的第一个元素。
- 名称在整个目录内必须是唯一的，它不区分大小写。
- 标号可以有 32 个字符 (最多)。
- 标号必须符合 IEC 命名惯例。
- 标号和后面的指令用冒号：隔开。
- 标号只能用在 “表达式”的起始处，否则在累加器内会出现未定义的数值。

例子：

```
start: LD A
       AND B
       OR C
       ST D
       JMP start
```

跳转的属性：

跳转的属性：
- 通过 JMP，可以实现到一个标号进行有条件或者无条件的跳转操作。
- JMP 可以与限定词 C 和 CN 一起使用 (仅限于累加器内容为 BOOL 数据类型的情况)。
- 跳转可以在程序内和 DFB 代码段内进行。
- 跳转只能在当前代码段内进行。

跳转可以有以下目标：
- EFB/DFB 调用的第一个 LD 指令 (参见 start2)，带有输入参数分配
- 一个正常的 LD 指令 (参见 start1)，
- 一个 CAL 指令，没有输入参数分配 (参见 start3)，
- 一个 JMP 指令 (参见 start4)，
- 一个指令表的结尾 (参见 start5)。

例子

```
start2: LD A
        ST counter.CU
        LD B
        ST counter.R
        LD C
        ST counter.PV
        CAL counter
        JMPCN start4
start1: LD A
        AND B
        OR C
        ST D
        JMPC start3
        LD A
        ADD E
        JMP start5
start3: CAL counter (
        CU:=A
        R:=B
        PV:=C )
        JMP start1
start4: JMPC start1
start5:
```

注释

描述

在 IL 编辑器中，注释总是以字符串（* 开头，以字符串 *) 结尾。在这些字符串中可以输入任何注释。
根据 IEC 61131-3，注释不能嵌套。如果要对注释进行嵌套，必须把它们明确激活。

14.2 调用基本功能，基本功能块，导出功能块和功能程序

介绍

概述

IL 编程语言中的调用基本功能，基本功能块，导出功能块和功能程序。

本节内容

本节包含以下主题：

基本功能调用

使用功能

基本功能以库的形式提供。功能的逻辑是用 C 语言编写的，不能在 IL 编辑器中修改。功能没有内部状态。如果输入值是相同的，那么功能执行的输出就都是一样的。比如说，针对两个数值相加的操作，功能执行都会给出相同的结果。对于一些基本功能来说，输入的数量是可以增加的。

基本功能只有一个返回值 (输出)。

参数

在向一个功能中传递数值，或者从一个功能向外传递数值时，需要用到 "输入"和一个 "输出"。它们称为形式参数。
当前进程状态中被传递了实际参数。它们称为实际参数。

下面各项可以用作功能输入的实际参数：
- 变量
- 地址
- 字面值

下面各项可以用作功能输出的实际参数：
- 变量
- 地址

实际参数的数据类型必须与形式参数的数据类型相匹配。唯一的例外是泛型实际参数，它的数据类型是由实际参数来决定的。
例外：在处理泛型形式参数 ANY_BIT 时，可以使用 INT 或者 DINT (不是 UINT 和 UDINT) 数据类型的实际参数。

它是 IEC 61131-3 的补充内容，必须被明确激活。

例子：
允许的：

```
AND (AnyBitParam := IntVar1, AnyBitParam2 := IntVar2)
```

不允许的：

```
AND_WORD (WordParam1 := IntVar1, WordParam2 := IntVar2)
```

(在这种情况下，必须使用 AND_INT。)

```
AND_ARRAY_WORD (ArrayInt, ...)
```

(在这种情况下，必须通过 INT_ARR_TO_WORD_ARR （...）；进行明确的类型转换。)

并不是所有形式参数都需要为其分配一个数值。您可以通过下面的表格来了解哪些形式参数类型必须赋值。

参数类型	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
输入	-	-	+	+	+	+	+	+
VAR_IN_OUT	+	+	+	+	+	+	/	+
输出	-	-	-	-	-	-	/	-
+ 需要实际参数								
- 不需要实际参数								
/ 不可用								

如果一个形式参数没有赋值，会使用初始值来执行功能。如果没有定义初始值，那么就会使用缺省值 (0)。

编程注意事项

在编程中应该注意以下事项：

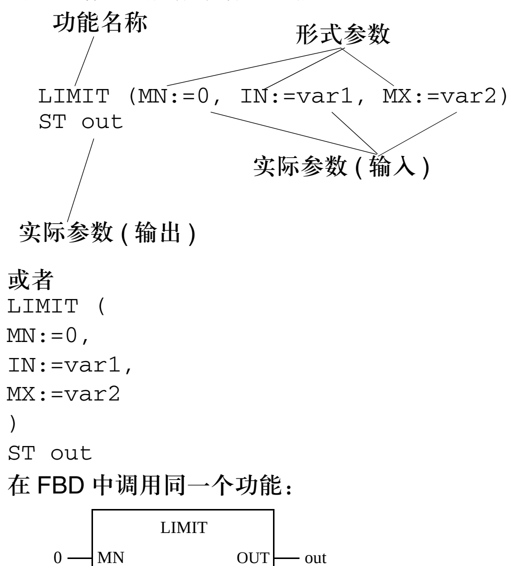
- 只有在输入 `EN`=1 或者未使用 `EN` 输入的情况下，才能执行功能 (请同时*参见 EN 和 ENO，406 页*)。
- 所有泛型功能都是重载的。这意味无论是否输入数据类型，都可以调用功能。
  比如
  ```
  LD i1
  ADD i2
  ST i3
  ```
  和
  ```
  LD i1
  ADD_INT i2
  ST i3
  ```
  是一样的
- 与 ST 不同， IL 中的功能不能嵌套。
- 有两种调用功能的方法：
  - 形式调用 (调用一个带有形式参数名称的功能)
  - 非形式调用 (调用一个不带形式参数名称的功能)

形式调用

在这种调用模式下 (通过形式参数名称进行调用)，功能是通过一个指令顺序进行调用的，该指令顺序以功能的名称开始，后面的括号中将数值 (实际参数) 赋给形式参数。形式参数的排列顺序**无关紧要**。实际参数可以紧跟在逗号之后。在执行了功能以后，结果会被加载到累加器中去，并可以通过 ST 保存起来。

这种调用可以使用 EN 和 ENO。

调用具有形式参数名称的功能：



或者

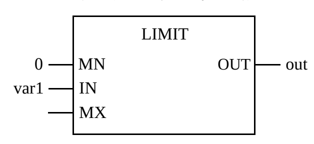
```
LIMIT (
MN:=0,
IN:=var1,
MX:=var2
)
ST out
```

在 FBD 中调用同一个功能：

在形式调用模式下，不需要对所有形式参数 (请同时*参见参数， 402 页*) 进行赋值。

```
LIMIT (MN:=0, IN:=var1)
ST out
```

在 FBD 中调用同一个功能：

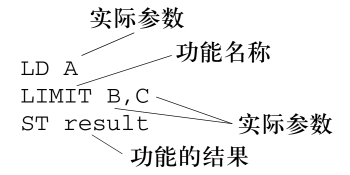


非形式调用

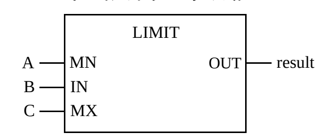
在这种调用模式下 (不通过形式参数名称进行调用)，功能是通过一个指令顺序进行调用的，该指令顺序以加载到累加器中的的一个实际参数开始，后面是功能名称以及一个可选的实际参数列表。实际参数的排列顺序**很重要**。实际参数可以紧跟在逗号之后。在执行了功能以后，结果会被加载到累加器中去，并可以通过 ST 保存起来。

这种调用**不能**使用 EN 和 ENO。

调用具有形式参数名称的功能：



在 FBD 中调用同一个功能：



> **注意：**请注意在进行非形式调用的时候，实际参数的列表**不能**放在括号中。在这种情况下，根据 IEC 61133-3 的要求，要把括号略去，以表明第一个实际参数不是列表的一部分。
>
> **针对**一个功能的**无效**非形式调用：
>
> ```
> LD A
> LIMIT (B,C)
> ```

如果要处理的数值 (第一个实际参数) 已经在累加器中了，就不需要使用加载命令了。

```
LIMIT B,C
ST result
```

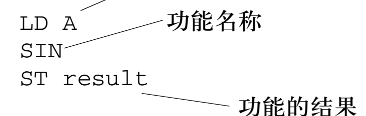
如果马上要对结果进行进一步处理，那么就不需要存储指令了。

```
LD A
LIMIT_REAL B,C
MUL E
```

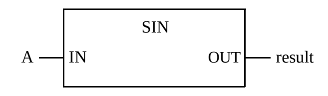
如果要执行的功能只有一个输入，那么功能的名称后面就不用带有一个实际参数列表了。

调用只有一个实际参数的功能：





在 FBD 中调用同一个功能：



EN 和 ENO

所有功能都可以配置一个 EN 输入和一个 ENO 输出。

如果 EN 的值等于 “0”，当功能被调用的时候，由功能所定义的算法不会执行，ENO 会被设为 “0”。

如果 EN 的值等于 “1”，当功能被调用的时候，由功能所定义的算法会执行。在这些算法成功执行以后， ENO 的值会被设为 “1”。如果在执行这些算法的时候出错，ENO 会置为 “0”。

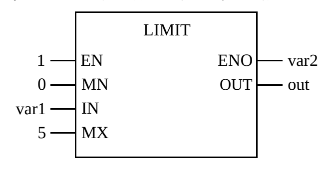
如果 ENO 置为 “0” （因为 EN=0 或者在执行的过程中出错），那么功能的输出会置为 “0”。

功能的输出行为与功能在被调用的时候是否不带有 EN/ENO 或者带有 EN=1 没有关系。

如果使用了 EN/ENO，那么功能调用必须是形式调用。

```
LIMIT (EN:=1, MN:=0, IN:=var1, MX:=5, ENO=>var2)
ST out
```

在 FBD 中调用同一个功能：



调用基本功能块和导出功能块

基本功能块

基本功能块具有内部状态。如果输入值相同，那么在不同的执行过程中，可能会有不同的输出值。比如，对于一个计数器来说，输出的数值是累加的。
功能块可以有多个输出值 (输出)。

导出功能块

导出功能块 (DFB) 与基本功能块的特性相同。用户可以通过 FBD， LD， IL，和 / 或 ST 编程语言来创建它们。

参数

在向功能块传递参数或者从功能块接收参数的时候，需要用到 “输入和输出”。它们称为形式参数。
当进程状态被传输到形式参数中。它们称为实际参数。
下面各项可以被用作功能块输入的实际参数：
- 变量
- 地址
- 字面值

下面各项可以被用作功能块输出的实际参数：
- 变量
- 地址

实际参数的数据类型必须与形式参数的数据类型相匹配。唯一的例外是泛型实际参数，它的数据类型是由实际参数来决定的。
例外：
在处理泛型形式参数 ANY_BIT 时，可以使用 INT 或者 DINT (不是 UINT 和 UDINT) 数据类型的实际参数。
它是 IEC 61131-3 的补充内容，必须被明确激活。
例子：
允许的：
```
AND (AnyBitParam := IntVar1, AnyBitParam2 := IntVar2)
```
不允许的：
```
AND_WORD (WordParam1 := IntVar1, WordParam2 := IntVar2)
```
(在这种情况下，必须使用 AND_INT)
```
AND_ARRAY_WORD (ArrayInt, ...)
```
(在这种情况下，必须通过 INT_ARR_TO_WORD_ARR (...)；进行明确的类型转换。)

并不是所有形式参数都需要为形式调用分配一个数值。您可以通过下面的表格来了解哪些形式参数类型必须赋值。

参数类型	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
EFB：输入	-	+	+	+	/	+	/	+
EFB: VAR_IN_OUT	+	+	+	+	+	+	/	+
EFB：输出	-	-	+	+	+	-	/	+
DFB：输入	-	+	+	+	/	+	/	+
DFB: VAR_IN_OUT	+	+	+	+	+	+	/	+
DFB：输出	-	-	+	/	/	-	/	+
+ 需要实际参数								
- 不需要实际参数								
/ 不可用								

如果一个形式参数没有赋值，会使用初始值来执行功能块。如果没有定义初始值，那么就会使用缺省值 (0)。

如果一个形式参数没有赋值，并且功能块 /DFB 多次实例化，那么后面的实例就会使用前面的数值来运行。

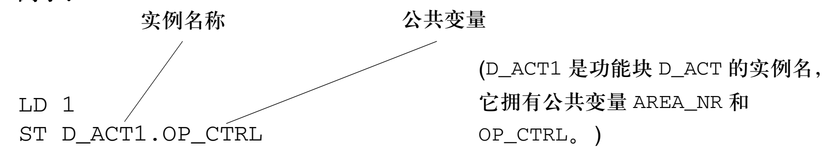
公共变量

除了 "输入 / 输出"以外，一些功能块还提供了公共变量。这些变量把统计数值 (不受进程影响的数值) 传递给功能块。它们用来设定功能块的参数。

公共变量是 IEC 61131-3 的补充内容。

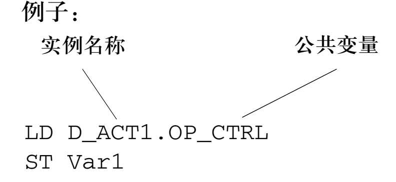
对公共变量的赋值是使用它们的初始值，或通过加载和保存指令来完成的。

例子：



公共变量通过功能块的实例名称和公共变量的名称来读入。

例子：



编程注意事项

在编程中应该注意以下事项：

- 只有在输入EN=1或者不使用输入EN的情况下，才能执行功能块(请同时*参见EN 和 ENO， 414 页*)。
- 对ANY或者ARRAY输出类型变量的赋值，必须使用=> 运算符(请同时参见*通过输入参数列表实现形式 CAL， 409 页*)。
  在功能块调用以外不能进行赋值操作。
  如果 SAH 功能块的输出 OUT 是 ANY 类型，那么下面的指令
  `My_Var := My_SAH.OUT`
  是无效的。
  指令
  `Cal My_SAH (OUT=>My_Var)`
  是有效的。
- 在使用 VAR_IN_OUT 变量 (参见 *VAR_IN_OUT 变量，414 页*) 的时候，可以应用一些特殊的条件。
- 对功能块的使用包含两个内容：
  - 声明 (参见*声明， 409 页*)
  - 调用功能块，
- 有四种调用功能块的方法：
  - 通过输入参数列表实现形式 CAL (请同时参见*通过输入参数列表实现形式 CAL， 409 页*) (通过形式参数名称进行调用)，在这种情况下，变量可以通过 => 运算符分配给输出。
  - 通过输入参数列表实现非形式 CAL(请同时参见*通过输入参数列表实现形式 CAL， 411 页*) (不通过形式参数名称进行调用)
  - CAL 和加载 / 保存 (参见 *CAL 和加载 / 保存输入参数， 412 页*) 输入参数
  - 使用输入运算符 (参见*使用输入运算符， 412 页*)
- 与通信 EFB 的实例只能被调用一次不同，功能块 /DFB 实例可以被多次调用 (参见*多重功能块实例调用， 414 页*)。

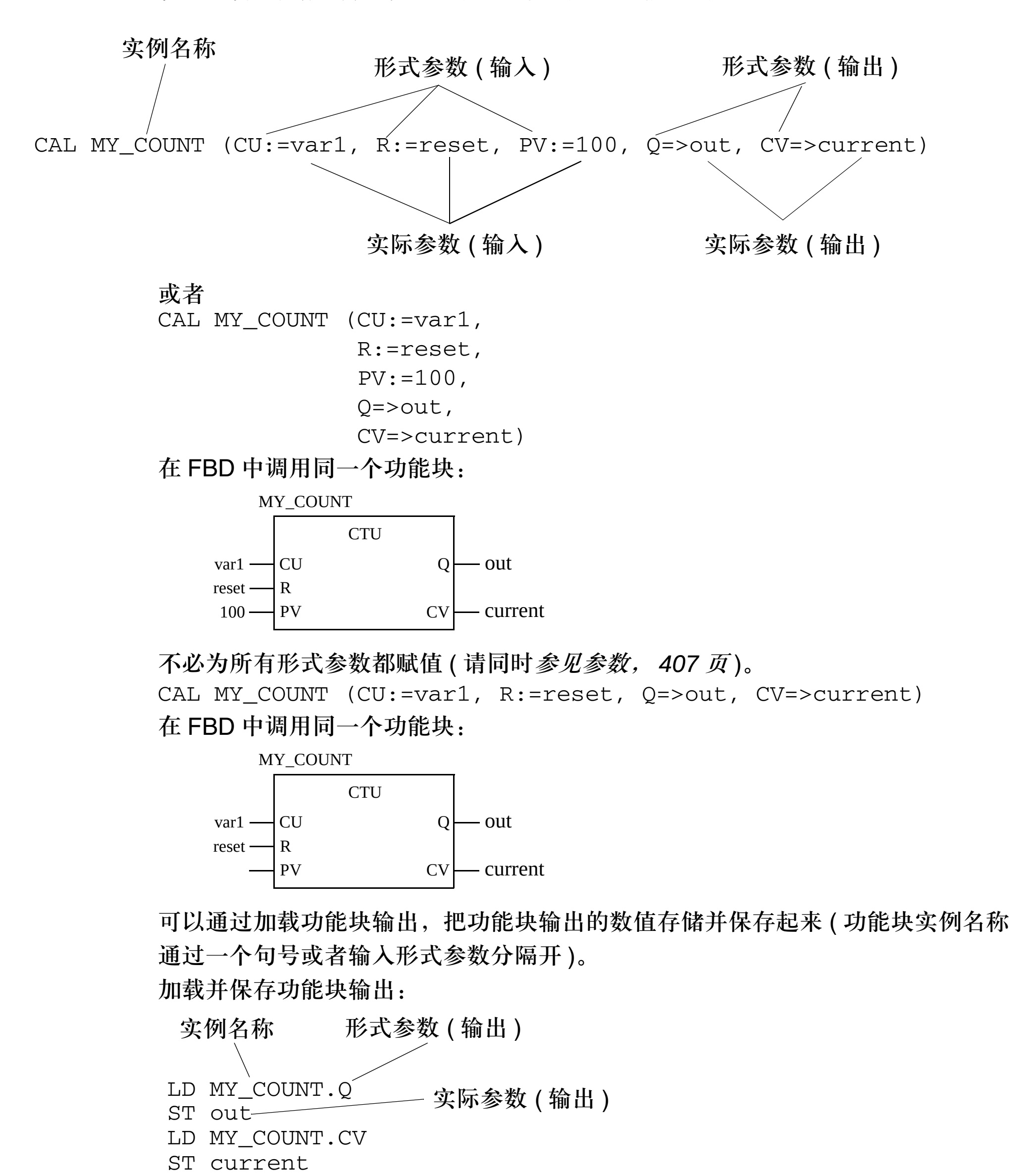
声明

在调用一个功能块之前，必须在变量编辑器中对它进行声明。

通过输入参数列表实现形式 CAL

在这种调用形式下 (通过形式参数名称进行调用)，功能块会通过一个 CAL 指令被调用，该指令跟在功能块的实例名称后面，并带有一个括号，在括号的列表中把实际参数赋给形式参数。输入形式参数的赋值操作是通过 := 赋值的格式来完成的，输出形式参数则是通过 => 赋值的格式来完成的。输入形式参数和输出形式参数排列的顺序**无关紧要**。实际参数的列表可以紧跟在逗号的后面。
这种调用可以使用 EN 和 ENO。

在通过输入参数列表实现形式 CAL 模式下的功能块调用：



或者

```
CAL MY_COUNT (CU:=var1,
              R:=reset,
              PV:=100,
              Q=>out,
              CV=>current)
```

在 FBD 中调用同一个功能块：

不必为所有形式参数都赋值 (请同时*参见参数， 407 页*)。

```
CAL MY_COUNT (CU:=var1, R:=reset, Q=>out, CV=>current)
```

在 FBD 中调用同一个功能块：

可以通过加载功能块输出，把功能块输出的数值存储并保存起来 (功能块实例名称通过一个句号或者输入形式参数分隔开)。

加载并保存功能块输出：

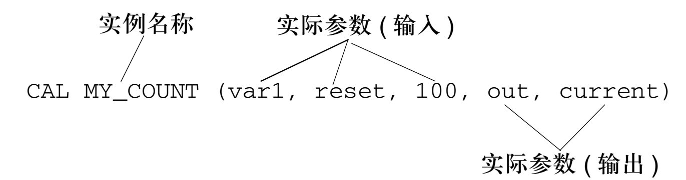

```
LD MY_COUNT.Q
ST out
LD MY_COUNT.CV
ST current
```

通过输入参数列表实现非形式 CAL

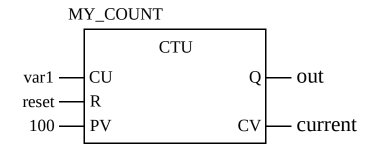
在这种调用形式下 (不通过形式参数名称进行调用)，功能块会通过一个 CAL 指令被调用，该指令跟在功能块的实例名称后面，并带有一个括号，括号中为用于输入和输出的实际参数列表。在功能块调用中实际参数的排列顺序很重要。

这种调用**不能**使用 EN 和 ENO。

在通过输入参数列表实现非形式 CAL 模式下的功能块调用：



在 FBD 中调用同一个功能块：



在非形式调用下，无需为所有的形式参数都赋值 (请同时*参见参数，407 页*)。

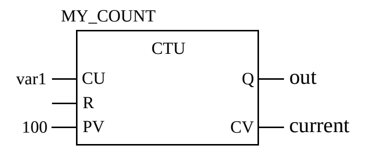
它是 IEC 61131-3 的补充内容，必须被明确激活。

如果要略过某个参数，可以使用一个空白参数栏。

通过空白参数栏进行调用：

```
CAL MY_COUNT (var1, , 100, out, current)
```

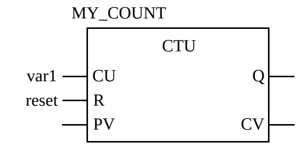
在 FBD 中调用同一个功能块：



如果在末尾处省略了形式参数，那么就不必使用空白参数栏。

```
MY_COUNT (var1, reset)
```

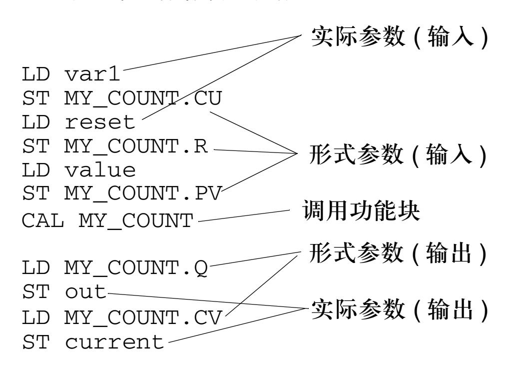
在 FBD 中调用同一个功能块



CAL 和加载 / 保存输入参数

功能块可以通过一个含有如下内容的指令表进行调用，该指令表包含加载实际参数，把实际参数保存到形式参数，以及 CAL 指令。加载和保存参数的顺序**无关紧要**。在第一个实际参数加载指令和调用功能块指令之间，只能使用针对当前正在配置的功能块的加载和保存指令。在该位置不能使用其他指令。
不必对所有的形式参数都进行赋值 (请同时参见*参数， 407 页*)。

CAL 和加载 / 保存输入参数：



使用输入运算符

功能块可以通过一个含有如下内容的指令表进行调用，该指令表包含加载实际参数，把实际参数保存到形式参数的操作，以及一个输入运算符。加载和保存参数的顺序**无关紧要**。
在第一个实际参数加载指令和功能块输入运算符之间，只能使用针对当前正在配置的功能块的加载和保存指令。在该位置不能使用其他指令。

这种调用**不能**使用 EN 和 ENO。

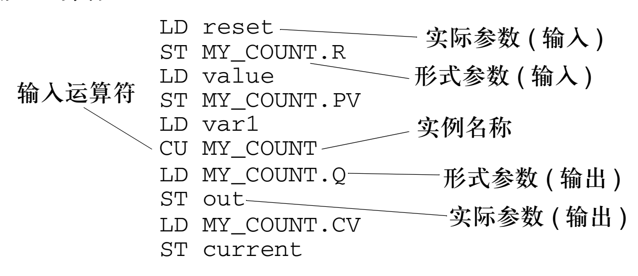
不必对所有形式参数都进行赋值 (请同时*参见参数， 407 页*)。

通过下面的表格，你可以了解各种功能块可以使用的输入运算符。不能使用附加的输入运算符。

输入运算符	FB 类型
S1, R	SR
S, R1	RS
CLK	R_TRIG
CLK	F_TRIG
CU, R, PV	CTU_INT, CTU_DINT, CTU_UINT, CTU_UDINT

输入运算符	FB 类型
CD, LD, PV	CTD_INT, CTD_DINT, CTD_UINT, CTD_UDINT
CU, CD, R, LD, PV	CTUD_INT, CTUD_DINT, CTUD_UINT, CTUD_UDINT
IN, PT	TP
IN, PT	TON
IN, PT	TOF

使用输入运算符：



调用没有输入的功能块

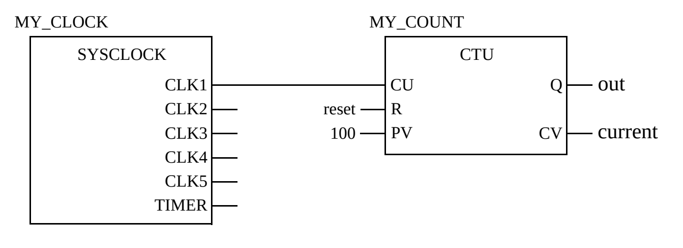
即使功能块没有输入，或者输入不需要参数化，功能块也应该在使用输出之前被调用，否则会传送输出的初始值，比如 “0”。

例子

调用 IL 编程语言的功能块：

```
CAL MY_CLOCK ()
CAL MY_COUNT (CU:=MY_CLOCK.CLK1, R:=reset, PV:=100)
LD MY_COUNT.Q
ST out
LD MY_COUNT.CV
ST current
```

在 FBD 中调用同一个功能块：



多重功能块实例调用

与通信 EFB 实例只能被调用一次不同，功能块 /DFB 实例可以被多次调用。

可以多次调用功能块 /DFB 实例，比如在下面的各种情况下：

- 如果功能块 /DFB 没有内部数值，或者后面的处理不需要它。
  在这种情况下，内存可以把它保存起来，因为功能块 /DFB 的代码只需加载一次，以后就可以多次调用同一个功能块 /DFB 实例。此时功能块 /DFB 会像一个“功能”那样被处理。
- 如果功能块 /DFB 带有一个内部数值，并且该数值会影响到各个程序段，比如说，计数器的数值应该在程序的各个部分增加。
  在这种情况下，调用同一个功能块 /DFB 时，不必为将来在程序的其他部分的处理保存当前结果。

EN 和 ENO

在所有功能块 /DFB 中都可以对一个 EN 输入和一个 ENO 输出进行配置。

如果在调用功能块 /DFB 的时候， EN 的数值等于 “0”，由功能块 /DFB 所定义的算法就不会执行， ENO 会置为 “0”。

如果在调用功能块 /DFB 的时候， EN 的数值等于 “1”，由功能块 /DFB 所定义的算法会执行。在成功执行这些算法以后， ENO 会置为 “1”。如果在执行这些算法的时候发生了错误， ENO 会置为 “0”。

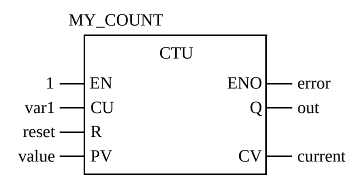
如果 ENO 被置 “0” (因为 EN=0 或者执行过程中发生错误)，功能块 /DFB 输出会保持上一个周期它们正确执行时的状态，

功能块 /DFB 的输出行为与 FFB 在被调用的时候是否不带有 EN/ENO 或者带有 EN=1 没有关系。

如果使用了 EN/ENO，功能块调用必须是形式调用。针对 ENO 的变量赋值操作必须通过 => 运算符来实现。

```
CAL MY_COUNT (EN:=1, CU:=var1, R:=reset, PV:=value,
           ENO=>error, Q=>out, CV=>current) ;
```

在 FBD 中调用同一个功能块：



VAR_IN_OUT 变量

功能块经常用于在一个输入读入变量 (输入变量)，处理该变量，以同一个变量名，更新并输出 (输出变量)。这种特殊类型的输入 / 输出变量也称为 VAR_IN_OUT 变量。

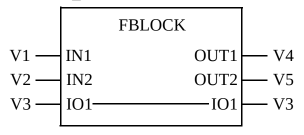
在使用带有 VAR_IN_OUT 变量的 FFB 时，需要注意以下特殊方面。

- 所有 VAR_IN_OUT 输入都必须分配一个变量。
- VAR_IN_OUT 输入**不能**分配字面值或者常数。
- VAR_IN_OUT 输出**不能**赋值。
- VAR_IN_OUT 变量不能在功能块调用以外使用。

调用一个带有 VAR_IN_OUT 变量的 IL 功能块：

```
CAL MY_FBLOCK(IN1:=V1, IN2:=V2, IO1:=V3,
               OUT1=>V4, OUT2=>V5)
```

在 FBD 中调用同一个功能块：



VAR_IN_OUT 变量**不能**在功能块调用以外使用。
因而，下面的功能块调用是**无效的**：

无效的调用，第一个例子：

LD V1	把变量 V1 加载到累加器。
CAL InOutFB	调用一个带有 VAR_IN_OUT 参数的功能块。 VAR_IN_OUT 参数的引用现在被加载到累加器中。
AND V2	累加器内容与变量 V2 的 AND 链接。 错误：不能执行操作，因为 VAR_IN_OUT 参数 (累加器内容) 不能在功能块调用的外部进行访问。

无效的调用，第二个例子：

LD V1	把变量 V1 加载到累加器。
AND InOutFB.inout	累加器内容与 VAR_IN_OUT 参数引用的 AND 链接。 错误：不能执行操作，因为 VAR_IN_OUT 参数不能在功能块调用的外部进行访问。

下面的功能块调用总是**有效的**：
有效的调用，第一个例子：

CAL InOutFB (IN1:=V1,inout:=V2	调用一个带有 VAR_IN_OUT 参数的功能块，在功能块调用中对实际参数进行赋值。

有效的调用，第二个例子：

LD V1	把变量 V1 加载到累加器。
ST InOutFB.IN1	把累加器的内容赋给功能块的参数 IN1。
CAL InOutFB(inout:=V2)	通过把实际参数（V2）赋给 VAR_IN_OUT 参数来调用功能块。

调用功能程序

功能程序

功能程序以库的形式提供。功能程序的逻辑是用 C 语言编写的，不能在 IL 编辑器中修改。功能程序和功能一样没有内部状态。如果输入值是相同的，那么功能程序执行的输出就都是一样的。比如说，针对两个数值相加的操作，功能程序执行都会给出相同的结果。
与功能不同，功能程序没有返回值，支持 VAR_IN_OUT 变量。

功能程序是 IEC 61131-3 的补充内容，必须被明确激活。

参数

在向一个功能程序内传递数值，或者从一个功能程序向外传递数值时，需要用到 “输入和输出”。它们称为形式参数。
当过程状态值传送给了形式参数。它们称为实际参数。
下面各项可以用作功能程序输入的实际参数：
- 变量
- 地址
- 字面值

下面各项可以用作功能程序输出的实际参数：
- 变量
- 地址

实际参数的数据类型必须与形式参数的数据类型相匹配。唯一的例外是泛型实际参数，它的数据类型是由实际参数来决定的。
例外：在处理泛型形式参数 ANY_BIT 时，可以使用 INT 或者 DINT （不是 UINT 和 UDINT) 数据类型的实际参数。

它是 IEC 61131-3 的补充内容，必须被明确激活。

例子：

允许的：
```
AND (AnyBitParam := IntVar1, AnyBitParam2 := IntVar2)
```
不允许的：
```
AND_WORD (WordParam1 := IntVar1, WordParam2 := IntVar2)
```
(在这种情况下，必须使用 AND_INT。)
```
AND_ARRAY_WORD (ArrayInt, ...)
```
(在这种情况下，必须通过 INT_ARR_TO_WORD_ARR （...）；进行明确的类型转换。)

并不是所有形式参数都需要为其分配一个数值。您可以通过下面的表格来了解哪些形式参数类型必须赋值。

参数类型	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
输入	-	-	+	+	+	+	+	+
VAR_IN_OUT	+	+	+	+	+	+	/	+
输出	-	-	-	-	-	-	/	+
+ 需要实际参数								
- 不需要实际参数								
/ 不可用								

如果一个形式参数没有分配任何数值，会使用初始值来执行功能。如果没有定义初始值，那么就会使用缺省值 (0)。

编程注意事项

在编程中应该注意以下事项：

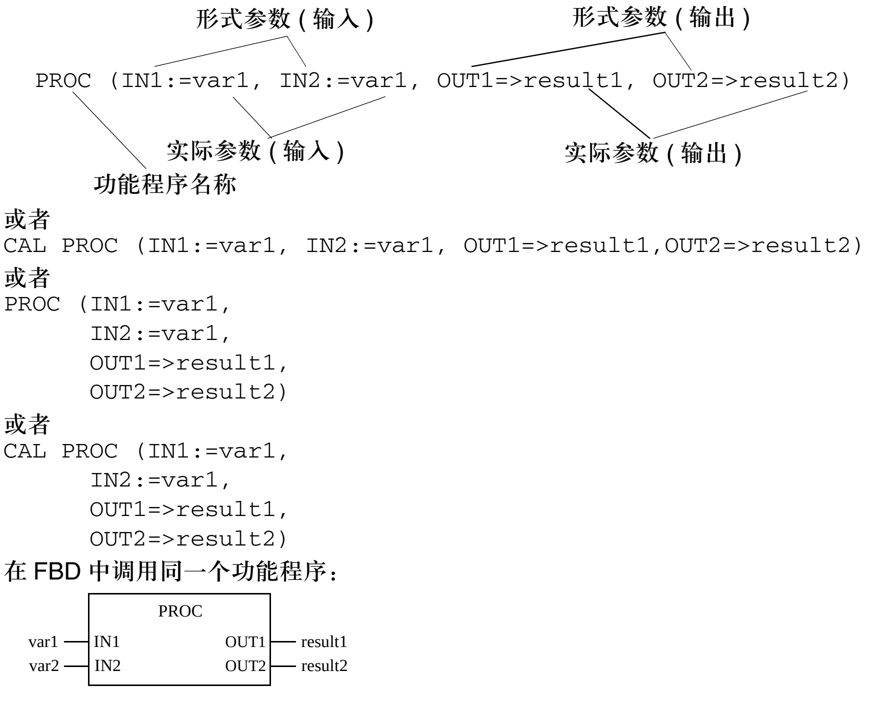
- 只有在输入 EN=1 或者不使用输入 EN 的情况下，才能执行功能程序 (请同时参见 EN 和 ENO，420 页)。
- 在使用 VAR_IN_OUT 变量的时候，要应用特殊的条件 (参见 VAR_IN_OUT 变量，421 页)。
- 有两种调用功能程序的方式：
  - 形式调用 (调用具有形式参数名称的功能)
  - 在这种情况下，变量可以通过 => 运算符分配给输出 (以缩短的形式调用功能块)。
  - 非形式调用 (调用一个不带形式参数名称的功能)

形式调用

在这种调用形式下 (通过形式参数名称进行调用)，功能程序会通过一个 CAL 指令被调用，该指令跟在功能程序的实例名称后面，并带有一个括号，在括号的列表中把实际参数赋给形式参数。输入形式参数的赋值操作是通过 := 赋值的格式来完成，输出形式参数则是通过 => 赋值的格式来完成。输入形式参数和输出形式参数排列的顺序**无关紧要**。实际参数的列表可以紧跟在逗号的后面。

这种调用可以使用 EN 和 ENO。

调用带有实际参数名称的功能程序：



或者

```
CAL PROC (IN1:=var1, IN2:=var1, OUT1=>result1,OUT2=>result2)
```

或者

```
PROC (IN1:=var1,
      IN2:=var1,
      OUT1=>result1,
      OUT2=>result2)
```

或者

```
CAL PROC (IN1:=var1,
      IN2:=var1,
      OUT1=>result1,
      OUT2=>result2)
```

在 FBD 中调用同一个功能程序：

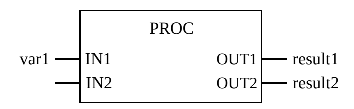
在形式调用中，不必为所有形式参数都赋值 (请同时*参见参数，416 页*)。

```
PROC (IN1:=var1, OUT1=>result1, OUT2=>result2)
```

或者

```
CAL PROC (IN1:=var1, OUT1=>result1, OUT2=>result2)
```

在 FBD 中调用同一个功能程序：

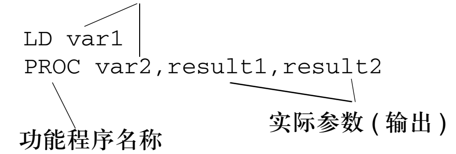


没有 CAL 指令的非形式调用

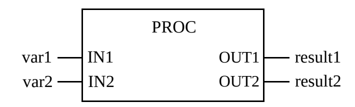
在这种调用模式下 (不通过形式参数名称进行调用)，功能程序是通过一个指令顺序进行调用的，该指令顺序以加载到累加器中的的一个实际参数开始，后面是功能程序名称以及一个可选的实际参数列表。实际参数的排列顺序**很重要**。实际参数列表可以紧跟在逗号之后。

这种调用**不能**使用 EN 和 ENO。

调用一个不带有形式参数名称的功能程序：



在 FBD 中调用同一个功能程序：



注意：请注意在进行非形式调用的时候，实际参数的列表**不能**放在括号中。在这种情况下，根据 IEC 61133-3 的要求，要把括号略去，以表明第一个实际参数不是列表的一部分。

针对一个功能的无效非形式调用：

```
LD A
LIMIT (B,C)
```

如果要处理的数值 (第一个实际参数) 已经在累加器中了，就不需要使用加载命令了。

```
EXAMP1 var2,result1,result2
```

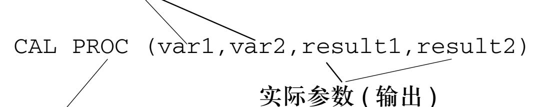
带有 CAL 指令的非形式调用

在这种调用模式下，功能程序是通过一个指令顺序进行调用的，该指令顺序以 CAL 指令开始，后面是功能名称以及一个输入和输出实际参数列表。实际参数的排列顺序很重要。实际参数列表可以紧跟在逗号之后。

这种调用**不能**使用 EN 和 ENO。

通过 CAL 指令调用带有形式参数名称的功能程序：

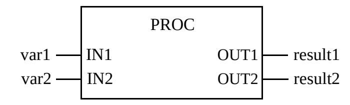




或者

```
CAL PROC (var1,
      var2,
      result1,
      result2)
```

在 FBD 中调用同一个功能程序：



注意：与不带 CAL 指令的非形式调用不同，在通过 CAL 指令进行非形式调用时，要处理的数值 (第一个实际参数) 没有没有明确加载到累加器，实际上它是实际参数列表的一部分。出于这个原因，在通过 CAL 指令进行非形式调用时，必须在括号内设定实际参数的列表。

EN 和 ENO

对所有功能程序来说，都可以对一个 EN 输入和一个 ENO 输出进行配置。

如果在调用功能程序的时候， EN 的数值等于 “0”，由功能程序所定义的算法就不会执行， ENO 会置为 “0”。

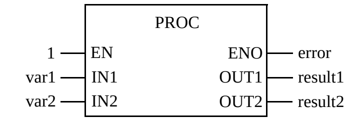
如果在调用功能程序的时候， EN 的数值等于 “1”，由功能程序所定义的算法会执行。在成功执行这些算法以后， ENO 会置为 “1”。如果在执行这些算法的时候发生了错误， ENO 会置为 “0”。

如果 ENO 置为 “0” (因为 EN=0 或者执行过程中发生错误)，功能程序输出会置为 “0”。

如果使用了 EN/ENO，功能程序调用必须是形式调用。针对 ENO 的变量赋值操作必须通过 => 运算符来实现。

```
PROC (EN:=1, IN1:=var1, IN2:=var2,
        ENO=>error, OUT1=>result1, OUT2=>result2) ;
```

在 FBD 中调用同一个功能程序：



VAR_IN_OUT 变量

功能程序经常用于在一个输入读入变量 (输入变量)，处理该变量，并以同一个变量名，更新并输出 (输出变量)。

这种特殊类型的输入 / 输出变量也称为 VAR_IN_OUT 变量。

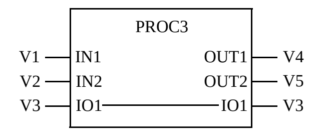
在使用带有 VAR_IN_OUT 变量的功能程序时，需要注意以下特殊方面。

- 所有 VAR_IN_OUT 输入都必须分配一个变量。
- VAR_IN_OUT 输入不能分配字面值或者常数。
- VAR_IN_OUT 输出不能赋值。
- VAR_IN_OUT 变量不能在功能块调用以外使用。

调用一个带有 VAR_IN_OUT 变量的 IL 功能块：

```
PROC3 (IN1:=V1, IN2:=V2, IO1:=V3,
          OUT1=>V4, OUT2=>V5) ;
```

在 FBD 中调用同一个功能程序：



VAR_IN_OUT 变量**不能**在功能程序调用以外使用。因而，下面的功能程序调用是无效的：

无效的调用，第一个例子：

LD V1	把变量 V1 加载到累加器。
CAL InOutProc	调用一个带有 VAR_IN_OUT 参数的功能程序。 VAR_IN_OUT 参数的引用现在被加载到累加器中。
AND V2	累加器内容与变量 V2 的 AND 链接。 错误：不能执行操作，因为 VAR_IN_OUT 参数 (累加器内容) 不能在功能程序调用的外部进行访问。

无效的调用，第二个例子：

LD V1	把变量 V1 加载到累加器。
AND InOutProc.inout	累加器内容与 VAR_IN_OUT 参数引用的 AND 链接。 错误：不能执行操作，因为 VAR_IN_OUT 参数不能在功能程序调用的外部进行访问。

无效的调用，第三个例子：

LD V1	把变量 V1 加载到累加器。
InOutFB V2	通过把实际参数（V2）赋给 VAR_IN_OUT 参数来调用功能程序。 **错误：**不能执行操作，因为对这种功能程序来说，只有 VAR_IN_OUT 参数被存储在累加器供进一步处理。

下面的功能程序总是**有效的**：

有效的调用，第一个例子：

CAL InOutProc (IN1:=V1,inout:=V2)	调用一个带有 VAR_IN_OUT 参数的功能程序，在功能程序调用中对实际参数进行形式赋值。

有效的调用，第二个例子：

InOutProc (IN1:=V1,inout:=V2)	调用一个带有 VAR_IN_OUT 参数的功能程序，在功能程序调用中对实际参数进行形式赋值。

有效的调用，第三个例子：

CAL InOutProc (V1,V2)	调用一个带有 VAR_IN_OUT 参数的功能程序，在功能程序调用中对实际参数进行形式赋值。

结构化文本 ST

15

介绍

概述

本章描述了遵循 IEC 61131 的结构化文本 ST 编程语言。

本章内容

本章包含以下各节：

15.1 关于结构化文本 ST 的常规信息

介绍

概述

本节给出了结构化文本 ST 的综述。

本节内容

本节包含以下主题：

关于结构化文本 ST 的常规信息

介绍

用户可以通过结构化文本 (ST) 编程语言来调用功能块，执行功能和赋值，有条件的执行指令，完成重复任务。

表达式

ST 编程语言通过 “表达式”来工作。表达式是由运算符和操作数构成的，它在执行的时候会返回一个数值。

运算符

运算符是在执行运算过程中所用到的符号。

操作数

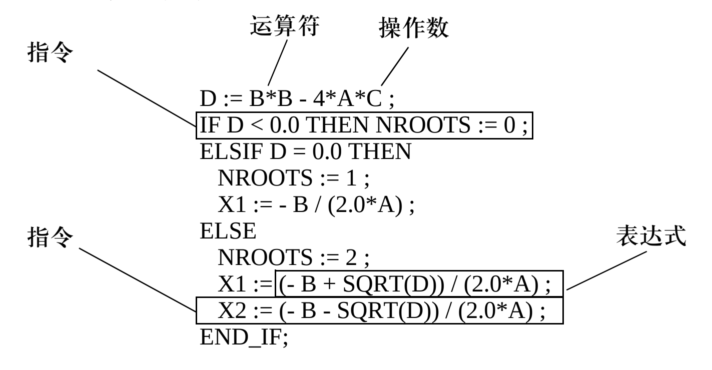
运算符用于操作数。操作数包括变量，字面值，功能以及 FFB 输入 / 输出，等等。

指令

指令用来把表达式的返回值赋给实际参数，并构建和控制表达式。

ST 代码段的示例

ST 代码段的示例：



代码段的大小

一个指令行的长度不能超过 300 个字符。
在编程环境中，没有限制 ST 代码段的长度。 ST 代码段的长度只受 PLC 内存的限制。

语法

标识符和关键字不区分大小写。

例外：不允许使用 空格和 tab 键

- 关键字，
- 字面值，
- 数值，
- 标识符，
- 变量 以及
- 限制符组合 [比如 (* 用于注释)]。

执行顺序

在表达式的运算中，要按照定义的运算符优先级顺序把运算符用于操作数 (参见*运算符表格，429 页*)。在表达式中，优先级最高的运算符会首先执行，然后是优先级低一级的运算符，如此下去，直到运算完成为止。在表达式中，具有同等优先级的运算符按照从左到右的顺序执行。可以通过括号来改变原来的执行顺序。

比如，如果 A， B， C 和 D 的值分别为 1， 2， 3 和 4，并按照如下方式运算：

```
A+B-C*D
```

那么结果就是 -9。

如果按照如下方式运算：

```
(A+B-C)*D
```

那么结果就是 0。

如果一个运算符包含两个操作数，那么在表达式中位于左面的操作数会首先被执行

```
SIN(A)*COS(B)
```

表达式 `SIN(A)` 会首先被执行，然后才是 `COS(B)`，这时才能给出计算的最终结果。

错误的行为

在执行一个表达式的时候，下面的情况被认为是发生了错误：

- 试图除 0
- 操作数不含有运算所需的正确数据类型
- 数字运算的结果超过了该数据类型的取值范围。

如果在进行运算的时候发生了错误，相关系统位 (%S) 会做出相应的置位 (如果正在使用的 PLC 支持它)。

遵循的 IEC 标准

关于 ST 编程语言所遵循的 IEC 标准，请参见 “遵循的 IEC 标准”。

操作数

介绍

操作数可以是：
- 一个地址
- 一个字面值
- 一个变量
- 一个多元素变量
- 多元素变量的一个元素
- 一个功能调用
- 一个 FFB 输出

数据类型

在一个指令中，要处理的操作数的数据类型必须是一致的。如果要处理不同类型的数据，必须首先进行类型转换。

在下面的例子中，整数变量 i1 被转换为一个实数变量，然后再与实数变量 r4 相加。

```
r3 := r4 + SIN(INT_TO_REAL(i1)) ;
```

这个规则有一个例外，那就是数据类型 TIME 的变量可以被数据类型 INT， DINT， UINT 或者 UDINT 的变量进行乘除运算。

允许的运算：
- `timeVar1 := timeVar2 / dintVar1;`
- `timeVar1 := timeVar2 intVar1;`
- `timeVar := 10 time#10s;`

该功能被 IEC 61131-3 列为 “不推荐使用”的内容。

直接使用地址

地址可以直接使用 (事先不需要给出声明)。在这种情况下，数据类型会直接分配给地址。分配过程是通过 “大前缀”来完成的。

下面的表格给出了各种大前缀：

大前缀 / 符号	例子	数据类型
没有前缀	%I10, %CH203.MOD, %CH203.MOD.ERR	BOOL
X	%M**X**20	BOOL
B	%Q**B**102.3	BYTE
W	%K**W**43	INT
D	%Q**D**100	DINT
F	%M**F**100	REAL

使用其他数据类型

如果要把其他数据类型分配为一个地址的缺省数据类型，那么必须给出明确的声明。这个变量声明可以借助变量编辑器来轻松地实现。地址的数据类型不能直接在 ST 代码段中进行声明 (比如说，声明 AT %MW1： UINT; 是不允许的)。

例如，下面的变量在变量编辑器中进行了声明：

```
UnlocV1: ARRAY [1..10] OF INT;
LocV1:   ARRAY [1..10] OF INT AT %MW100;
LocV2:   TIME AT %MW100;
```

这样一来，下面的调用就拥有了正确的语法：

```
%MW200 := 5;
UnlocV1[2] := LocV1[%MW200];
LocV2      := t#3s;
```

访问数组变量

在访问数组变量（ARRAY）的时候，在索引项中只能使用 INT， DINT， UINT 和 UDINT 类型的字面值和变量。

如果一个 ARRAY 元素的下限是负值，那么它的索引就可以是负值。

例子：使用字段变量

```
var1[i] := 8 ;
var2.otto[4] := var3 ;
var4[1+i+j*5] := 4 ;
```

运算符

介绍

运算符是一个符号，它用来：
- 执行一个算术运算或者
- 执行一个逻辑运算或者
- 一个功能编辑 (调用)。

运算符是通用的，也就是说，它们会与操作数的数据类型自动匹配。

运算符表格

运算符按照优先级的顺序来执行，请同时参见*执行顺序， 426 页*。

ST 编程语言的运算符：

运算符	含义	优先级级别	可能的操作数	描述

`()`	使用括号：	1 (最高)	表达式	括号用来改变运算符的执行顺序。
				比如：如果操作数 A， B， C 和 D 的值分别为 1， 2， 3， 4，那么
				`A+B-C*D`
				结果为 -9
				`(A+B-C)*D`
				结果为 0。

FUNCNAME (实际参数列表)	功能编辑 (调用)	2	表达式，字面值，变量，地址 (所有数据类型)	功能处理用来执行功能 (参见*调用基本功能， 453 页*)。
`-`	否定	3	INT， DINT， UINT， UDINT 或 REAL 数据类型的表达式， 字面值，变量，地址	在否定的过程中， - 符号把操作数的正负符号变为相反 **比如：**在下面的例子中，如果 IN1 为 4，那么 OUT 为 -4。 `OUT := - IN1;`
NOT	求反	3	BOOL， BYTE， WORD 或 DWORD 数据类型的表达式，字面值，变量，地址	NOT 运算符对操作数进行逐位的求反操作。 **比如：**在下面的例子中，如果 IN1 为 1100110011，那么 OUT 就为 0011001100。 `OUT := NOT IN1 ;`
`**`	求幂	4	REAL (底数) 和 INT， DINT， UINT， UDINT 或 REAL (指数) 数据类型的表达式，字面值，变量，地址	求幂运算符 ** 使用第二个操作数 (指数)，对第一个操作数 (底数) 进行求幂操作。 **比如：**在下面的例子中，如果 IN1 是 5.0， IN2 是 4.0，那么 OUT 就是 625.0。 `OUT := IN1 ** IN2;`

运算符	含义	优先级级别	可能的操作数	描述
*	乘法	5	INT， DINT， UINT， UDINT， REAL 或者 TIME 数据类型的表达式，字面值，变量，地址	乘法运算符 * 把第一个操作数的值与第二个操作数的值相乘。 **比如：**在下面的例子中，如果 IN1 是 5.0，IN2 是 4.0，那么 OUT 就是 20.0。 OUT ：= IN1 * IN2 ;
/	除法	5	INT， DINT， UINT， UDINT 或者 REAL 数据类型的表达式，字面值，变量，地址	除法运算符 / 把第一个操作数的值除以第二个操作数的值。 **比如：**在下面的例子中，如果 IN1 是 20.0， IN2 是 5.0，那么 OUT 就是 4.0。 OUT ：= IN1 / IN2 ;
MOD	求模	5	INT， DINT， UINT 或者 UDINT 数据类型的表达式，字面值， 变量，地址	MOD 运算符把把第一个操作数的值除以第二个操作数的值，余数 (模数) 作为结果。 **比如：**在下面的例子中 ● 如果 IN1 是 7， IN2 是 2， OUT 就是 1 ● 如果 IN1 是 7，IN2 是 -2，OUT 就是 1 ● 如果 IN1 是 -7，IN2 是 2，OUT 就是 -1 ● 如果 IN1 是 -7，IN2 是 -2，OUT 就是 -1 OUT ：= IN1 MOD IN2 ;
+	加法	6	INT， DINT， UINT， UDINT， REAL 或者 TIME 数据类型的表达式，字面值，变量，地址	加法运算符 + 把第一个操作数的值和第二个操作数的值相加。 **比如：**在下面的例子中 如果 IN1 是 7， IN2 是 2， OUT 就是 9 OUT ：= IN1 + IN2 ;
-	减法	6	INT， DINT， UINT， UDINT， REAL 或者 TIME 数据类型的表达式，字面值，变量，地址	减法运算符 – 从第一个操作数的值中减去第二个操作数的值。 **比如：**在下面的例子中，如果 IN1 是 10，IN2 是 4， OUT 就是 6。 OUT ：= IN1 - IN2 ;
<	小于比较	7	BOOL， BYTE， INT， DINT， UINT， UDINT， REAL， TIME， WORD， DWORD， STRING， DT， DATE 或者 TOD 数据类型的表达式，字面值， 变量，地址	< 运算符把第一个操作数的值与第二个操作数的值进行比较。如果第一个操作数小于第二个操作数，结果就是布尔 1。如果第一个操作数大于或等于第二个操作数，结果就是布尔 0。 **比如：**在下面的例子中，如果 IN1 小于 10， OUT 就是 1，否则 OUT 就是 0。 OUT ：= IN1 < 10 ;

运算符	含义	优先级级别	可能的操作数	描述
>	大于比较	7	BOOL，BYTE，INT，DINT，UINT，UDINT，REAL，TIME，WORD，DWORD，STRING，DT，DATE 或者 TOD 数据类型的表达式，字面值，变量，地址	> 运算符把第一个操作数的值与第二个操作数的值进行比较。如果第一个操作数大于第二个操作数，结果就是布尔 1。如果第一个操作数小于或等于第二个操作数，结果就是布尔 0。 **比如：**在下面的例子中，如果 IN1 大于 10，OUT 就是 1，否则 OUT 就是 0。 `OUT := IN1 > 10 ;`
<=	小于等于比较	7	BOOL，BYTE，INT，DINT，UINT，UDINT，REAL，TIME，WORD，DWORD，STRING，DT，DATE 或者 TOD 数据类型的表达式，字面值，变量，地址	<= 运算符把第一个操作数的值与第二个操作数的值进行比较。如果第一个操作数小于或等于第二个操作数，结果就是布尔 1。如果第一个操作数大于第二个操作数，结果就是布尔 0。 **比如：**在下面的例子中，如果 IN1 小于或等于 10，OUT 就是 1，否则 OUT 就是 0。 `OUT := IN1 <= 10 ;`
>=	大于等于比较	7	BOOL，BYTE，INT，DINT，UINT，UDINT，REAL，TIME，WORD，DWORD，STRING，DT，DATE 或者 TOD 数据类型的表达式，字面值，变量，地址	>= 运算符把第一个操作数的值与第二个操作数的值进行比较。如果第一个操作数大于或等于第二个操作数，结果就是布尔 1。如果第一个操作数小于第二个操作数，结果就是布尔 0。 **比如：**在下面的例子中，如果 IN1 大于或等于 10，OUT 就是 1，否则 OUT 就是 0。 `OUT := IN1 >= 10 ;`
=	等于	8	BOOL，BYTE，INT，DINT，UINT，UDINT，REAL，TIME，WORD，DWORD，STRING，DT，DATE 或者 TOD 数据类型的表达式，字面值，变量，地址	= 运算符把第一个操作数的值与第二个操作数的值进行比较。如果第一个操作数等于第二个操作数，结果就是布尔 1。如果第一个操作数不等于第二个操作数，结果就是布尔 0。 **比如：**在下面的例子中，如果 IN1 等于 10，OUT 就是 1，否则 OUT 就是 0。 `OUT := IN1 = 10 ;`

运算符	含义	优先级级别	可能的操作数	描述
<>	不等于	8	BOOL， BYTE， INT， DINT， UINT， UDINT， REAL， TIME， WORD， DWORD， STRING， DT， DATE 或者 TOD 数据类型的表达式，字面值， 变量，地址	<> 运算符把第一个操作数的值与第二个操作数的值进行比较。如果第一个操作数不等于第二个操作数，结果就是布尔 1。如果第一个操作数等于第二个操作数，结果就是布尔 0。 **比如：**在下面的例子中，如果 IN1 不等于 10， OUT 就是 1，否则 OUT 就是 0。 OUT : = IN1 <> 10 ;
&	逻辑与	9	BOOL， BYTE， WORD 或 DWORD 数据类型的表达式，字面值，变量，地址	& 运算符为操作数之间建立逻辑与链接关系，对 BYTE， WORD 和 DWORD 数据类型来说，这个链接是逐位进行的。 **比如：**在下面的例子中，如果 IN1， IN2 和 IN3 为 1，那么 OUT 就是 1。 OUT : = IN1 & IN2 & IN3 ;
AND	逻辑与	9	BOOL， BYTE， WORD 或 DWORD 数据类型的表达式，字面值，变量，地址	AND 运算符为操作数之间建立逻辑与链接关系，对 BYTE， WORD 和 DWORD 数据类型来说，这个链接是逐位进行的。 **比如：**在下面的例子中，如果 IN1， IN2 和 IN3 为 1，那么 OUT 就是 1。 OUT : = IN1 AND IN2 AND IN3 ;

运算符	含义	优先级级别	可能的操作数	描述

XOR	逻辑异或	10	BOOL， BYTE， WORD 或 DWORD 数据类型的表达式，字面值，变量，地址	XOR 运算符为操作数之间建立逻辑异或链接关系，对 BYTE， WORD 和 DWORD 数据类型来说，这个链接是逐位进行的。
				比如：在下面的例子中，如果 IN1 和 IN2 不相等，那么 OUT 就是 1。如果 IN1 和 IN2 具有同样的状态 (同为 0 或 1)，那么 OUT 就是 0。
				`OUT : = IN1 XOR IN2 ;`
				如果链接的操作数超过两个，那么如果 1 状态的数量是奇数，结果就是 1，如果 1 状态的数量是偶数，结果就是 0。
				比如：在下面的例子中，如果 1 个或 3 个操作数为 1，OUT 就是 1，如果 2 个或 4 个操作数为 1， OUT 就是 0。
				`OUT := IN1 XOR IN2 XOR IN3 XOR IN4 ;`

OR	逻辑或	11 (最低)	BOOL， BYTE， WORD 或 DWORD 数据类型的表达式，字面值，变量，地址	OR 运算符为操作数之间建立逻辑或链接关系，对 BYTE， WORD 和 DWORD 数据类型来说，这个链接是逐位进行的。 **比如：**在下面的例子中，如果 IN1， IN2 或者 IN3 为 1，那么 OUT 就是 1。 `OUT : = IN1 OR IN2 OR IN3 ;`

15.2 指令

介绍

概述

本章描述了结构化文本 ST 编程语言的指令。

本节内容

本节包含以下主题：

指令

描述

指令是 ST 编程语言的 “命令”。指令必须通过分号来中止。
在一行中可以包含多个指令 (用分号隔开)。
一个指令如果只包含一个分号，就是一个空指令 (参见*空指令；， 449 页*)。

赋值

介绍

在赋值操作中，一个或多个变量的当前值被表达式的运算结果所取代。
一个赋值指令包括位于左侧的变量规范，变量后面的赋值运算符：=，以及位于运算符后面，需要进行计算的表达式。
两侧 (赋值运算符左侧和右侧) 的变量必须具有相同的数据类型。
数组的情况比较特殊。在经过了明确说明以后，可以使用两个长度不同的数组。

把一个变量的值赋给另外一个变量

可以通过赋值操作把一个变量的值赋给另外一个变量。
比如，可以通过下面的指令

```
A ：= B ;
```

把变量 A 的值更换为变量 B 的当前值。如果 A 和 B 是基本数据类型，那么 B 的值会传递给 A。如果 A 和 B 是导出数据类型，那么 B 所有元素的值都会传递给 A。

把一个字面值赋给一个变量

可以通过赋值操作把一个字面值赋给一个变量。
比如，可以通过下面的指令

```
C ：= 25 ;
```

把 25 赋给变量 C。

把一个运算的值赋给一个变量

可以通过赋值操作把一个运算的结果赋给一个变量。
比如，可以通过下面的指令

```
X ：= (A+B-C)*D ;
```

把运算 (A+B-C)*D 的结果赋给变量 X。

把一个 FFB 的值赋给一个变量

可以通过赋值操作把一个功能或者功能块的返回值赋给一个变量。

比如，可以通过下面的指令

```
B : = MOD(C, A) ;
```

调用 MOD (求模) 功能，把计算结果赋给变量 B。

可以通过下面的指令

```
A : = MY_TON.Q ;
```

把 MY_TON 功能块的 Q 输出（TON 功能块的实例）的值赋给 A 变量 (这不是一个功能块调用) 。

多重赋值

多重赋值是 IEC 61131-3 的补充内容，必须被明确激活。

即使已经被激活，多重赋值也不能用于以下场合：

- 用在功能块调用的参数列表中
- 用在对结构化变量进行初始化的元素列表中

下面的指令

```
X : = Y : = Z
```

是可以的。

下面的指令

```
FB(in1 : = 1, In2 : = In3 : = 2) ;
```

以及

```
strucVar : = (comp1 : = 1, comp2 : = comp3 : = 2) ;
```

是不允许的。

在数组和字 / 双字变量之间进行赋值

只有在事先进行了类型转换的情况下，才能在数组和字 / 双字变量之间进行赋值，比如：

```
MOVE_INT_AREBOOL(IN:=%MW20,OUT=>%M1:16) ;
```

有以下可用的转换功能 (常规库)，数组族：

- MOVE_BOOL_AREBOOL
- MOVE_WORD_ARWORD
- MOVE_DWORD_ARDWORD
- MOVE_INT_ARINT
- MOVE_DINT_ARDINT
- MOVE_REAL_ARREAL

选择指令 IF...THEN...END_IF

描述

如果使用了 IF 指令，那么只有在相关的布尔表达式的值为 1 (真) 时，才能执行一个或者一组指令。如果条件为 0 (假)，该指令或指令组就不会执行。
THEN 指令用来表明条件结束，指令开始。
END_IF 指令用来表明指令结束。

> **注意：**可以对无限多个 IF… THEN… ELSE… END_IF 指令进行嵌套，以生成复合选择指令。

IF...THEN... END_IF 的例子

条件可以用一个布尔变量来表达。
如果 FLAG 是 1，指令会执行；如果 FLAG 是 0，它们就不会执行。

```
IF FLAG THEN
   C:=SIN(A) * COS(B) ;
   B:=C - A ;
END_IF ;
```

也可以借助一个返回一个布尔结果的运算来表达条件。如果 A 大于 B，指令会执行；如果 A 小于或等于 B，它们就不会执行。

```
IF A>B THEN
   C:=SIN(A) * COS(B) ;
   B:=C - A ;
END_IF ;
```

IF NOT...THEN... END_IF 的例子

条件可以通过 NOT 求反 (当 FLAG 为 0 时执行这两个指令)。

```
IF NOT FLAG THEN
   C:=SIN_REAL(A) * COS_REAL(B) ;
   B:=C - A ;
END_IF ;
```

请同时参见

ELSE (参见*选择指令 ELSE，439 页*)
ELSIF (参见*选择指令 ELSIF...THEN，440 页*)

选择指令 ELSE

描述

ELSE 指令总是跟在一个 IF...THEN 指令， ELSIF...THEN 指令或者 CASE 指令的后面。

如果 ELSE 指令跟在一个 IF 或者 ELSIF 指令的后面，那么只有在 IF 和 ELSIF 指令相关的布尔表达式为 0(假) 时，才会执行指令或者指令组。如果 IF 和 ELSIF 指令的条件为 1(真)，指令或者指令组不会执行。

如果 ELSE 指令跟在 CASE 指令的后面，那么只有在任何标号都不包含选择符的数值时，指令或者指令组才会被执行。如果某一个标号含有选择符的数值，那么指令或者指令组就不会执行。

> **注意：**可以对无限多个 IF...THEN...ELSE...END_IF 指令进行嵌套，以生成复合选择指令。

ELSE 的例子

```
IF A>B THEN
   C:=SIN(A) * COS(B) ;
   B:=C - A ;
ELSE
   C:=A + B ;
   B:=C * A ;
END_IF ;
```

请同时参见

IF (参见*选择指令 IF...THEN...END_IF， 438 页*)
ELSIF (参见*选择指令 ELSIF...THEN， 440 页*)
CASE (参见*选择指令 CASE...OF...END_CASE， 441 页*)

选择指令 ELSIF...THEN

描述

ELSIF 指令总是跟在一个 IF...THEN 指令的后面。如果使用了 ELSIF 指令，那么只有当 IF 指令相关布尔表达式的值为 0 (假)，并且 ELSIF 指令相关布尔表达式的值为 1 (真) 时，才会执行一个指令或者一组指令。如果 IF 指令的条件是 1 (真)，或者 ELSIF 指令的条件是 0 (假)，那么指令或者指令组就不会执行。 THEN 指令表明 ELSIF 条件结束，指令开始。

注意：可以对无限多个 IF...THEN...ELSIF...THEN...END_IF 指令进行嵌套，以生成复合选择指令。

ELSIF...THEN 的例子

```
IF A>B THEN
   C:=SIN(A) * COS(B) ;
   B:=SUB(C,A) ;
ELSIF A=B THEN
   C:=ADD(A,B) ;
   B:=MUL(C,A) ;
END_IF ;
```

比如 nested 指令

```
IF A>B THEN
     IF B=C THEN
         C:=SIN(A) * COS(B) ;
     ELSE
         B:=SUB(C,A) ;
     END_IF ;
ELSIF A=B THEN
     C:=ADD(A,B) ;
     B:=MUL(C,A) ;
ELSE
     C:=DIV(A,B) ;
END_IF ;
```

请同时参见

IF (参见*选择指令 IF...THEN...END_IF*， *438 页*)
ELSE (参见*选择指令 ELSE*， *439 页*)

选择指令 CASE...OF...END_CASE

描述

CASE 指令含有一个 INT 数据类型表达式（“选择符”）和指令组的一个列表。每一个都有一个标号，该标号包含一个或多个整数（INT，DINT，UINT，UDINT）或者整数范围值。标号含有选择符运算值的一组指令会执行，除此以外，所有的其他指令都不会执行。 OF 指令表示标号开始。

在 CASE 指令内可以使用一个 ELSE 指令，如果所有标号都不含有选择符的数值，那么就会执行 ELSE 指令。

END_CASE 指令表示指令结束。

CASE...OF... END_CASE 的例子

CASE...OF...END_CASE 的例子

选择符

标号

```
CASE SELECT OF
   1,5:   C:=SIN(A) * COS(B) ;
   2:     B:=C - A ;
   6..10: C:=C * A ;
ELSE
   B:=C * A ;
   C:=A / B ;
END_CASE ;
```

请同时参见

ELSE （参见*选择指令 ELSE， 439*）

循环指令 FOR...TO...BY...DO...END_FOR

描述

如果事先能够确定重复的次数，那么就应该使用 FOR 指令，否则应该使用 WHILE（参见*循环指令 WHILE...DO...END_WHILE，444 页*）或者 REPEAT（参见*循环指令 REPEAT...UNTIL...END_REPEAT，445 页*）指令。 FOR 指令会重复一个指令执行，直到遇到 END_FOR 指令为止。重复的次数由起始值，最终值和控制变量来决定。

控制变量，起始值和最终值必须具有相同的数据类型（DINT 或 INT)。

控制变量，起始值和最终值可以通过一个循环的指令来更改。这是 IEC 61131-3 的补充内容。

通过 FOR 指令，控制变量的数值会从起始值累加到最终值。在缺省情况下，累加值为 1。如果需要使用另外一个累加值，可以指定一个明确的累加值 (变量或常数)。在每一次新的循环之前，都会检查控制变量的数值，如果该数值超过了起始值和最终值的范围，就会跳出该循环体。

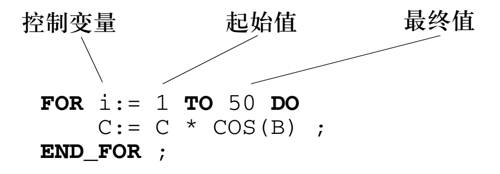
在第一次进行循环之前，会进行相关检查，以确保控制变量是从起始值向最终值变化。如果情况不是这样 (比如起始值≤最终值以及负增长)，那么循环就不会继续下去。控制变量不能在循环体的外面进行定义。

DO 指令表示循环定义结束，指令开始。

可以通过 EXIT 指令来提前中止循环。 END_FOR 指令表示指令结束。

例子：累加量为 1 的 FOR 指令

累加量为 1 的 FOR 指令



累加量不为 1 的 FOR 指令

如果要使用的累加量不为 1，可以通过 BY 进行定义。累加量，控制变量，起始值和最终值必须具有相同的数据类型 (DINT 或 INT)。判断处理方向 (正向，反向) 的标准是 BY 表达式的符号。如果这个表达式是正的，循环会正向运行；如果它是负的，循环会反向运行。

例子：在两步内正向计数

在两步内正向计数


```
FOR i:= 1 TO 10 BY 2 DO (* BY > 0 : Forwards.loop *)
  C:= C * COS(B) ; (* Loop is 5 x executed *)
END_FOR ;
```

例子：反向计数

反向计数

```
FOR i:= 10 TO 1 BY -1 DO (* BY < 0 : Backwards.loop *)
  C:= C * COS(B) ; (* Instruction is executed 10 x *)
END_FOR ;
```

例子："一次"循环

在下面的例子中，因为起始值 = 最终值，所以循环只执行了一次。在这种情况下，累加量的正负没有关系。

```
FOR i:= 10 TO 10 DO (* Unique Loop *)
  C:= C * COS(B) ;
END_FOR ;
```

或者

```
FOR i:= 10 TO 10 BY -1 DO (* Unique Loop *)
  C:= C * COS(B) ;
END_FOR ;
```

例子：临界循环

如果例子中的累加量 j > 0，指令会执行。

如果 j < 0，指令不会执行，因为起始值只能以≥ 0 的方式向最终值变化。

如果 j = 0，指令会执行，并因为永远也达不到最终值而造成死循环。

```
FOR i:= 1 TO 10 BY j DO
  C:= C * COS(B) ;
END_FOR ;
```

循环指令 WHILE...DO...END_WHILE

描述

如果使用了 WHILE 指令，那么一个指令序列会重复执行，直到相关的布尔表达式为 0 (假)。如果表达式从一开始就为假，那么指令组就不会执行。

DO 指令表示循环的定义结束，指令开始。

可以使用 EXIT 来提前中止循环。

END_WHILE 指令表示指令结束。

在下面的场合下不能使用 WHILE，因为它会造成死循环，从而使程序失败：

- WHILE 不能用于进程间的同步化操作，比如用作一个带有外部定义的中止条件的“等待循环”。
- WHILE 不能用在一个算法中，因为无法保证循环结束条件的完成或者 EXIT 指令的执行。

WHILE...DO...END_WHILE 的例子

```
x := 1
WHILE x <= 100 DO
      x := x + 4;
END_WHILE ;
```

请同时参见

EXIT （参见*循环指令 EXIT，446 页*）

循环指令 REPEAT...UNTIL...END_REPEAT

描述

如果使用了 Repeat 指令，会重复执行一个指令顺序 (至少一次)，直到相关的布尔条件为 1(真)。
UNTIL 指令给出中止条件。
可以使用 EXIT 来提前中止循环。
END_Repeat 指令表示指令结束。
在下面的场合下不能使用 Repeat，因为它会造成死循环，从而使程序失败：

- REPEAT 不能用于进程间的同步化操作，比如用作一个带有外部定义的中止条件的 “等待循环”。
- REPEAT 不能用在一个算法中，因为无法保证循环结束条件的完成或者 EXIT 指令的执行。

REPEAT...UNTIL ...END_REPEAT 的例子

```
x := -1
REPEAT
          x := x + 2
   UNTIL x >= 101
END_REPEAT ;
```

请同时参见

EXIT (参见 *循环指令 EXIT， 446 页*)

循环指令 EXIT

描述

EXIT 指令用来在满足结束条件以前，提前中止循环指令（FOR， WHILE， REPEAT）。
如果 EXIT 指令在一个嵌套的循环内，程序会离开最里层循环（EXIT 所处的位置 ），然后执行跟在该循环后面的第一个指令（END_FOR， END_WHILE 或者 END_REPEAT）。

EXIT 的例子

如果 FLAG 的值为 0，在执行指令以后， SUM 的值为 15。如果 FLAG 的值为 1，在执行指令以后， SUM 的值为 6。

```
SUM : = 0 ;
FOR I := 1 TO 3 DO
   FOR J := 1 TO 2 DO
      IF FLAG=1 THEN EXIT;
      END_IF ;
      SUM := SUM + J ;
   END_FOR ;
   SUM := SUM + I ;
END_FOR
```

请同时参见

CASE(参见*选择指令 CASE...OF...END_CASE， 441 页*)
WHILE(参见*循环指令 WHILE...DO...END_WHILE， 444 页*)
REPEAT(参见*循环指令 REPEAT...UNTIL...END_REPEAT， 445 页*)

子程序调用

子程序调用

一个子程序调用包括子程序代码段的名称以及其后的一个空参数列表。

子程序调用没有返回值。

要调用的子程序必须和调用的 ST 代码段位于同一个任务内。

子程序也可以从子程序内部进行调用。

比如：

```
Subroutine Name() ;
```

子程序调用是 IEC 61131-3 的补充内容，必须被明确激活。

在 SFC 动作代码段中，只有在激活了多令牌操作的情况下，才能进行子程序调用。

RETURN 指令

描述

每一个子程序和 DFB（导出功能块）在被处理完毕以后都会退出，也就是说，返回到被调用的主程序。

如果要子程序 /DFB 提早离开，可以通过 RETURN 指令使其返回主程序。

RETURN 只能用于子程序或者 DFB 中。它们不能用在主程序内。

空指令；

描述

一个单独的分号 ; 代表一个空指令。

比如，

```
IF x THEN ; ELSE ..
```

在这个例子中， THEN 指令后面跟着一个空指令。这表示只要 IF 条件为 1，程序就会马上跳出 IF 指令。

标记和跳转

介绍

标记是跳转的目标。
在 ST 中的跳转和标记是 IEC 61131-3 的附加内容，必须被明确激活。

标记的属性

标记的属性：
- 标记必须是一行中的第一个元素。
- 标记只能出现在第一次序指令的前面 (不在循环内)。
- 名称在整个目录内必须是唯一的，它不区分大小写。
- 标记可以有 32 个字符 (最多)。
- 标记必须符合 IEC 命名惯例。
- 标记必须和后面的指令以冒号：隔开。

跳转的属性

跳转的属性
- 可以在程序和 DFB 代码段内进行跳转。
- 跳转只能在当前代码段内进行。

例子

```
IF var1 THEN
   JMP START;
:
:
START: ...
```

注释

描述

在 ST 编辑器中，注释总是以字符串 (* 开头，以字符串 *) 结尾。在这些字符串中可以输入任何注释。除了关键字，字面值，标识符和变量以外，注释可以在 ST 编辑器的任何位置输入。
根据 IEC 61131-3，注释不能嵌套。如果要对注释进行嵌套，必须把它们明确激活。

15.3 调用基本功能，基本功能块，导出功能块和功能程序

介绍

概述

在 ST 编程语言中调用基本功能，基本功能块，导出功能块和功能程序。

本节内容

本节包含以下主题：

调用基本功能

基本功能

基本功能以库的形式提供。功能的逻辑是用 C 语言编制的，不能在 ST 编辑器中修改。功能没有内部状态。如果输入值是相同的，那么功能执行的输出就都是一样的。比如说，针对两个数值相加的操作，功能执行都会给出相同的结果。对于一些基本功能来说，输入的数量可以增加到两个以上。

基本功能只有一个返回值 (输出)。

参数

在向一个功能内传递数值，或者从一个功能向外传递数值时，需要用到 “输入”和一个 “输出”。它们称为形式参数。

当进程状态值传递给形式参数。它们称为实际参数。

下面各项可以用作功能输入的实际参数：

- 变量
- 地址
- 字面值
- ST 表达式

下面各项可以用作功能输出的实际参数：

- 变量
- 地址

实际参数的数据类型必须与形式参数的数据类型相匹配。唯一的例外是泛型实际参数，它的数据类型是由实际参数来决定的。

例外：在处理泛型形式参数 ANY_BIT 时，可以使用 INT 或者 DINT （不是 UINT 和 UDINT）数据类型的实际参数。

它是 IEC 61131-3 的补充内容，必须被明确激活。

例子：

允许的：

```
AND (AnyBitParam := IntVar1, AnyBitParam2 := IntVar2);
```

不允许的：

```
AND_WORD (WordParam1 := IntVar1, WordParam2 := IntVar2);
```

(在这种情况下，必须使用 AND_INT。)

```
AND_ARRAY_WORD (ArrayInt, ...);
```

(在这种情况下必须通过 INT_ARR_TO_WORD_ARR (...) ；进行明确的类型转换。)

并不是所有形式参数都需要赋值。您可以通过下面的表格来了解哪些形式参数类型必须赋值。

参数类型	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
输入	-	-	+	+	+	+	+	+
VAR_IN_OUT	+	+	+	+	+	+	/	+
输出	-	-	-	-	-	-	/	-
+ 需要实际参数								
- 不需要实际参数								
/ 不可用								

如果一个形式参数没有赋值，会使用初始值来执行功能。如果没有定义初始值，那么就会使用缺省值 (0)。

编程注意事项

在编程中应该注意以下事项：

- 所有泛型功能都是重载的。这意味无论是否输入数据类型，都可以调用功能。
  比如
  `i1 := ADD (i2, 3);`
  与
  `i1 := ADD_INT (i2, 3);` 是一样的。
- 功能可以嵌套 (请同时参见嵌套功能， 456 页。
- 只有在输入 EN=1 或者不使用输入 EN 的情况下，才能执行功能 (请同时参见 *EN 和 ENO， 457 页*)。
- 有两种调用功能的方法：
  - 形式调用 (调用具有形式参数名称的功能)
  - 非形式调用 (调用一个不带形式参数名称的功能)

形式调用

在形式调用模式下 (通过形式参数名称进行调用)，调用包含输出的实际参数，参数后面带有赋值指令：= 和功能名称，并带有一个含有赋值列表的括号 (把实际参数赋给形式参数)。在功能调用中，形式参数的排列顺序无关紧要。

这种调用可以使用 EN 和 ENO。

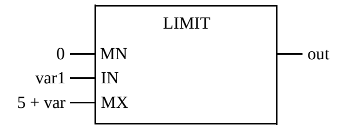
调用具有形式参数名称的功能：

实际参数（输出）　　　　形式参数

```
out:=LIMIT (MN:=0, IN:=var1, MX:=5 + var) ;
```

功能名称　　　　实际参数（输入）

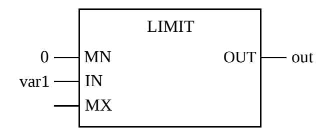
在 FBD 中调用同一个功能：



在形式调用模式下，不必为所有形式参数都赋值 (请同时参见*参数， 453 页*)。

```
out:=LIMIT (MN:=0, IN:=var1) ;
```

在 FBD 中调用同一个功能：

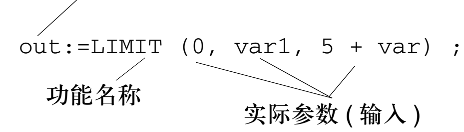


非形式调用

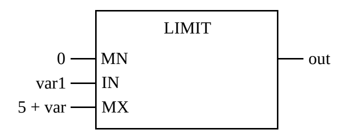
在非形式调用模式下 (不通过形式参数名称进行调用)，调用包含输出的实际参数，参数后面带有赋值指令符号：= 和功能名称，并带有一个含有输入实际参数列表的括号。在功能调用中，实际参数的排列顺序**很重要**。

这种调用**不能**使用 EN 和 ENO。

调用一个不带形式参数名称的功能：



在 FBD 中调用同一个功能：



在非形式调用模式下，不必为所有形式参数都赋值 (请同时参见*参数， 453 页*)。

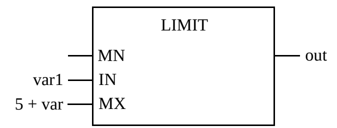
它是 IEC 61131-3 的补充内容，必须被明确激活。

可以用一个空参数栏来略过一个参数。

通过空参数栏进行调用：

```
out:=LIMIT ( ,var1, 5 + var) ;
```

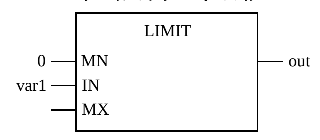
在 FBD 中调用同一个功能：



如果形式参数在结尾处被省略，那么可以不使用空参数栏。

```
out:=LIMIT (0, var1) ;
```

在 FBD 中调用同一个功能：



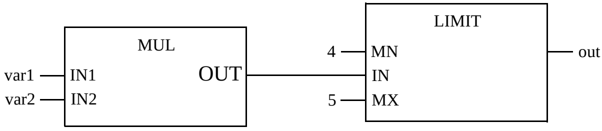
嵌套功能

在调用一个功能的过程中，可以进一步进行功能调用。嵌套的层数没有限制。

数组功能的嵌套调用：

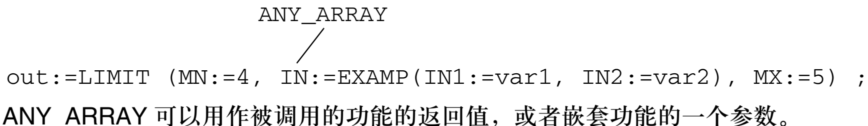
```
out:=LIMIT (MN:=4, IN:=MUL(IN1:=var1, IN2:=var2), MX:=5) ;
```

在 FBD 中调用同一个功能：



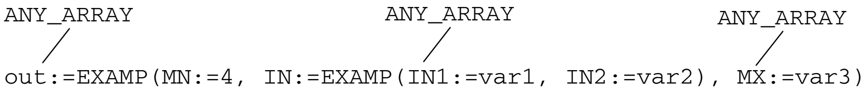
在功能调用**内部不能**使用返回值为 ANY_ARRAY 数据类型的功能。

通过 ANY_ARRAY 进行的**不被认可**的嵌套：



ANY_ARRAY 可以用作被调用的功能的返回值，或者嵌套功能的一个参数。

通过 ANY_ARRAY 进行的**被认可**的嵌套：



EN 和 ENO

在所有功能中都可以对一个 EN 输入和一个 ENO 输出进行配置。

如果在调用功能的时候， EN 的数值等于 “0”，由功能所定义的算法就不会执行， ENO 会置为 “0”。

如果在调用功能的时候， EN 的数值等于 “1”，由功能所定义的算法会执行。在成功执行这些算法以后， ENO 会置为 “1”。如果在执行这些算法的时候发生了错误， ENO 会置为 “0”。

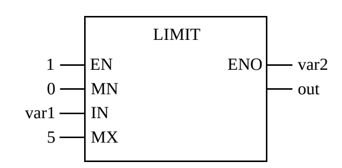
如果 ENO 置为 “0” (因为 EN=0 或者执行过程中发生错误)，那么功能的输出会置为 “0”。

功能的输出行为与功能在被调用的时候是否不带有 EN/ENO 或者带有 EN=1 没有关系。

如果使用了 EN/ENO，那么功能调用必须是形式调用。

```
out:=LIMIT (EN:=1, MN:=0, IN:=var1, MX:=5, ENO=>var2) ;
```

在 FBD 中调用同一个功能：



调用基本功能块和导出功能块

基本功能块

基本功能块具有内部状态。如果输入值相同，那么在不同的执行过程中，可能会有不同的输出值。比如，对于一个计数器来说，输出的数值是累加的。
功能块可以有多个输出值 (输出)。

导出功能块

导出功能块 (DFB) 与基本功能块的特性相同。用户可以通过 FBD， LD， IL，和 / 或 ST 编程语言来创建它们。

参数

在向功能块传递参数或者从功能块接收参数的时候，需要用到 “输入和输出”。它们称为形式参数。
当进程状态传输到形式参数中。它们称为实际参数。
下面各项可以被用作功能块输入的实际参数：

- 变量
- 地址
- 字面值

下面各项可以被用作功能块输出的实际参数：

- 变量
- 地址

实际参数的数据类型必须与形式参数的数据类型相匹配。唯一的例外是泛型实际参数，它的数据类型是由实际参数来决定的。

例外：在处理泛型形式参数 ANY_BIT 时，可以使用 INT 或者 DINT （不是 UINT 和 UDINT）数据类型的实际参数。

它是 IEC 61131-3 的补充内容，必须被明确激活。

例子：

允许的：

```
AND (AnyBitParam := IntVar1, AnyBitParam2 := IntVar2);
```

不允许的：

```
AND_WORD (WordParam1 := IntVar1, WordParam2 := IntVar2);
```

(在这种情况下，必须使用 AND_INT。)

```
AND_ARRAY_WORD (ArrayInt, ...);
```

(在这种情况下，必须通过 INT_ARR_TO_WORD_ARR (...) ；进行明确的类型转换。)

并不是所有形式参数都需要赋值。您可以通过下面的表格来了解哪些形式参数类型必须赋值。

参数类型	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
EFB：输入	-	+	+	+	/	+	/	+
EFB: VAR_IN_OUT	+	+	+	+	+	+	/	+
EFB：输出	-	-	+	+	+	-	/	+
DFB：输入	-	+	+	+	/	+	/	+
DFB: VAR_IN_OUT	+	+	+	+	+	+	/	+
DFB：输出	-	-	+	/	/	-	/	+
+ 需要实际参数								
- 不需要实际参数								
/ 不可用								

如果一个形式参数没有赋值，会使用初始值来执行功能块。如果没有定义初始值，那么就会使用缺省值 (0)。

如果一个形式参数没有赋值，并且功能块 /DFB 多次实例化，那么后面的实例就会使用前面的数值来运行。

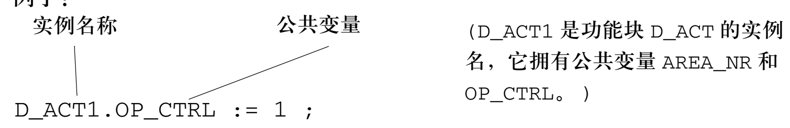
公共变量

除了 “输入 / 输出”以外，一些功能块还提供了公共变量。这些变量把统计数值 (不受进程影响的数值) 传递给功能块。它们用来设定功能块的参数。

公共变量是 IEC 61131-3 的补充内容。

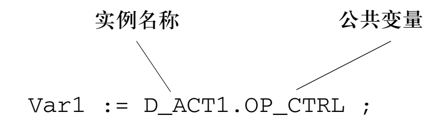
对公共变量赋值的过程是借助它们的初始值或者赋值操作来完成的。

例子：



公共变量通过功能块的实例名称和公共变量的名称来读入。

例子：



编程注意事项

在编程中应该注意以下事项：

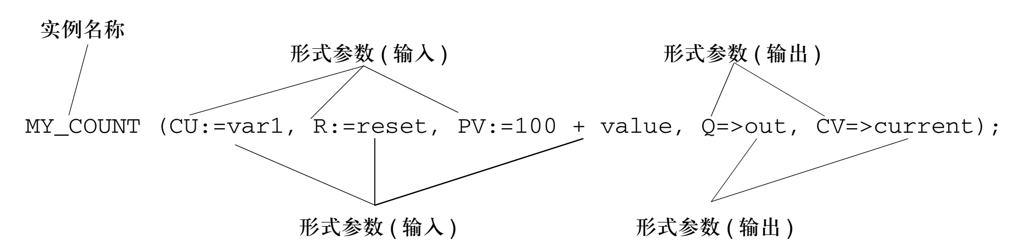
- 只有在输入 EN=1 或者不使用输入 EN 的情况下，才能执行功能块（请同时参见 *EN 和 ENO，463 页*）。
- 对 ANY 或者 ARRAY 输出类型变量的赋值，必须使用 => 运算符 (请同时参见*形式调用， 460 页*)。
  在功能块调用以外不能进行赋值操作。
  如果 My_SAH 功能块的输出 OUT 是 ANY 类型，那么下面的指令
  `My_Var : = My_SAH.OUT;`
  是无效的。
  下面的指令
  `Cal My_SAH (OUT=>My_Var);`
  是有效的。
- 在使用 VAR_IN_OUT 变量 (参见 *VAR_IN_OUT 变量，464 页*) 的时候，要使用一些特殊的条件。
- 在 ST 中对功能块的使用包含两个内容：
  - 声明 (参见*声明， 460 页*)
  - 调用功能块，
- 有两种调用功能块的方法：
  - 形式调用 (参见*形式调用， 460 页*) (调用具有形式参数名称的功能)
    通过这种方式，变量可以借助 => 运算符赋给输出。
  - 非形式调用 (参见*非形式调用， 461 页*) (不通过形式参数名称调用)
- 与通信 EFB 的实例只能被调用一次不同，功能块 /DFB 实例可以被多次调用 (参见*多重功能块实例调用， 463 页*)。

声明

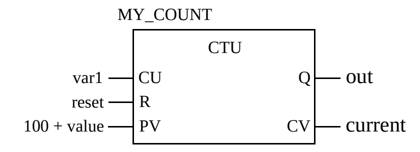
在调用一个功能块之前，必须在变量编辑器中进行声明。

形式调用

在形式调用模式下 (通过形式参数名称进行调用)，功能块会通过一个来自功能块实例名称的指令顺序来调用，并带有一个括号，在括号的列表中把实际参数赋给形式参数。输入形式参数的赋值操作是通过：= 赋值的格式来完成的，输出形式参数则是通过 => 赋值的格式来完成的。输入形式参数和输出形式参数排列的顺序**无关紧要**。
这种调用可以使用 EN 和 ENO。
通过形式参数名称调用功能块：



在 FBD 中调用同一个功能块：



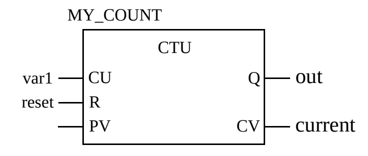
对功能块输出进行赋值，也可以首先输入一个实际参数名称，然后使用赋值指令：=，最后是功能块实例名称和功能块输出形式参数，它们之间用句号分开。
比如

```
MY_COUNT (CU:=var1, R:=reset, PV:=100 + value);
Q := out ;
CV := current ;
```

不必为所有形式参数都赋值 (请同时参见*参数，458 页*)。

```
MY_COUNT (CU:=var1, R:=reset, Q=>out, CV=>current);
```

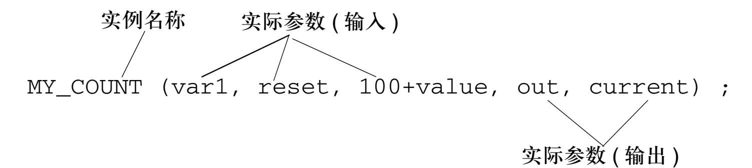
在 FBD 中调用同一个功能块：



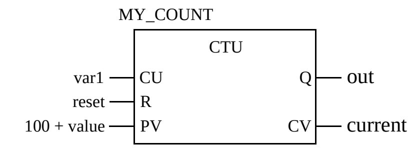
非形式调用

在非形式调用模式下 (不通过形式参数名称进行调用)，功能块会通过一个来自功能块实例名称的指令来调用，并带有一个含有输入和输出实际参数列表的括号。实际参数的排列顺序**很重要**。
这种调用**不能**使用 EN 和 ENO。

不通过形式参数名称调用功能块：



在 FBD 中调用同一个功能块：



在非形式调用下，无需为所有的形式参数都赋值 (请同时参见*参数， 458 页*)。这个规则不适用于 VAR_IN_OUT 变量，带有动态长度的输入参数以及 ANY 类型的输出。它们必须分配一个变量。

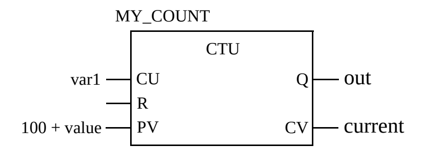
它是 IEC 61131-3 的补充内容，必须被明确激活。

如果要略过某个参数，可以使用一个空白参数栏。

通过空白参数栏进行调用：

```
MY_COUNT (var1, , 100 + value, out, current) ;
```

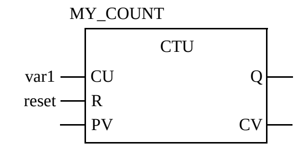
在 FBD 中调用同一个功能块：



如果形式参数在结尾处被省略，那么可以不使用空参数栏。

```
MY_COUNT (var1, reset) ;
```

在 FBD 中调用同一个功能块



调用没有输入的功能块

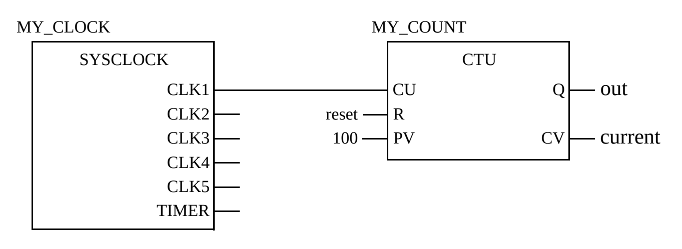
即使功能块没有输入，或者输入不需要参数化，功能块也应该在使用输出之前被调用，否则会传送输出的初始值，比如 “0”。

例子

调用 ST 编程语言的功能块：

```
MY_CLOCK () ;
MY_COUNT (CU:=MY_CLOCK.CLK1, R:=reset, PV:=100,
          Q=>out, CV=>current) ;
```

在 FBD 中调用同一个功能块：



多重功能块实例调用

与通信 EFB 实例只能被调用一次不同，功能块 /DFB 实例可以被多次调用。

可以多次调用功能块 /DFB 实例，比如在下面的各种情况下：

- 如果功能块 /DFB 没有内部数值，或者后面的处理不需要它。
  在这种情况下，内存会把它保存起来，因为功能块 /DFB 的代码只需加载一次，可以通过多次调用同一个功能块 /DFB 实例。这时功能块 /DFB 会像一个 “功能”那样被处理。
- 如果功能块 /DFB 带有一个内部数值，并且该数值会影响到各个程序段，比如说，计数器的数值应该在程序的各个部分增加。
  在这种情况下，调用同一个功能块 /DFB 时，不必为将来在程序的其他部分所进行的处理保存当前结果。

EN 和 ENO

在所有功能块 /DFB 中都可以对一个 EN 输入和一个 ENO 输出进行配置。

如果在调用功能块 /DFB 的时候， EN 的数值等于 “0”，由功能块 /DFB 所定义的算法就不会执行， ENO 会置为 “0”。

如果在调用功能块 /DFB 的时候，EN 的数值等于“1”，由功能块 /DFB 所定义的算法会执行。在成功执行这些算法以后， ENO 会置为 “1”。如果在执行这些算法的时候发生了错误， ENO 会置为 “0”。

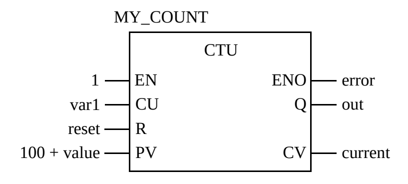
如果 ENO 置为 “0” (因为 EN=0 或者执行过程中发生错误)，功能块 /DFB 输出会保持上一个周期它们正确执行时的状态，

功能块 /DFB 的输出行为与 FFB 在被调用的时候是否不带有 EN/ENO 或者带有 EN=1 没有关系。

如果使用了 EN/ENO，功能块调用必须是形式调用。针对 ENO 的变量赋值操作必须通过 => 运算符来实现。

```
MY_COUNT (EN:=1, CU:=var1, R:=reset, PV:=100 + value,
          ENO=>error, Q=>out, CV=>current) ;
```

在 FBD 中调用同一个功能块：



VAR_IN_OUT 变量

功能块经常用来在一个输入读入变量 (输入变量)，处理该变量，并使用同一个变量名更新输出 (输出变量)。

这种特殊类型的输入 / 输出变量也称为 VAR_IN_OUT 变量。

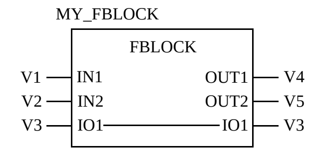
在使用带有 VAR_IN_OUT 变量的 FFB 时，需要注意以下特殊方面。

- 所有 VAR_IN_OUT 输入都必须分配一个变量。
- VAR_IN_OUT 输入不能分配字面值或者常数。
- VAR_IN_OUT 输出**不能**赋值。
- VAR_IN_OUT 变量**不能**在功能块调用以外使用。

调用一个带有 VAR_IN_OUT 变量的 ST 功能块：

```
MY_FBLOCK(IN1:=V1, IN2:=V2, IO1:=V3, OUT1=>V4, OUT2=>V5);
```

在 FBD 中调用同一个功能块：



VAR_IN_OUT 变量**不能**在功能块调用以外使用。

因而，下面的功能块调用是**无效的**：

无效的调用，第一个例子：

`InOutFB.inout := V1;`	把变量 V1 赋给一个 VAR_IN_OUT 参数。 错误：不能执行操作，因为 VAR_IN_OUT 参数不能在功能块调用的外部进行访问。

无效的调用，第二个例子：

`V1 := InOutFB.inout;`	把一个 VAR_IN_OUT 参数赋给 V1 变量。 错误：不能执行操作，因为 VAR_IN_OUT 参数不能在功能块调用的外部进行访问。

下面的功能块调用总是**有效的**：

有效的调用，第一个例子：

`InOutFB (inout:=V1);`	调用一个带有 VAR_IN_OUT 参数的功能程序，在功能程序调用中用实际参数进行赋值。

有效的调用，第二个例子

`InOutFB (V1);`	调用一个带有 VAR_IN_OUT 参数的功能程序，在功能程序调用中用实际参数进行赋值。

功能程序

功能程序

功能程序以库的形式提供。功能程序的逻辑是用 C 语言编写的，不能在 ST 编辑器中修改。功能程序和功能一样没有内部状态。如果输入值是相同的，那么所功能程序执行的输出就都是一样的。比如说，针对两个数值相加的操作，功能程序执行都会给出相同的结果。

与功能不同，功能程序没有返回值，支持 VAR_IN_OUT 变量。

功能程序是 IEC 61131-3 的补充内容，必须被明确激活。

参数

在向一个功能程序中传递数值，或者从一个功能程序向外传递数值时，需要用到“输入和输出”。它们称为形式参数。

当前进程状态被传递给实际参数。它们称为实际参数。

下面各项可以用作功能程序输入的实际参数：

- 变量
- 地址
- 字面值
- ST 表达式

下面各项可以用作功能程序输出的实际参数：

- 变量
- 地址

实际参数的数据类型必须与形式参数的数据类型相匹配。唯一的例外是泛型实际参数，它的数据类型是由实际参数来决定的。

例外：在处理泛型形式参数 ANY_BIT 时，可以使用 INT 或者 DINT （不是 UINT 和 UDINT）数据类型的实际参数。

它是 IEC 61131-3 的补充内容，必须被明确激活。

例子：

允许的：

```
AND (AnyBitParam := IntVar1, AnyBitParam2 := IntVar2);
```

不允许的：

```
AND_WORD (WordParam1 := IntVar1, WordParam2 := IntVar2);
```

(在这种情况下，必须使用 AND_INT。)

```
AND_ARRAY_WORD (ArrayInt, ...);
```

(在这种情况下，必须通过 INT_ARR_TO_WORD_ARR （...）；进行明确的类型转换。)

并不是所有形式参数都需要赋值。您可以通过下面的表格来了解哪些形式参数类型必须赋值。 .

参数类型	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
输入	-	-	+	+	+	+	+	+
VAR_IN_OUT	+	+	+	+	+	+	/	+
输出	-	-	-	-	-	-	/	+
+ 需要实际参数								
- 不需要实际参数								
/ 不可用								

如果一个形式参数没有赋值，会使用初始值来执行功能。如果没有定义初始值，那么就会使用缺省值 (0)。

编程注意事项

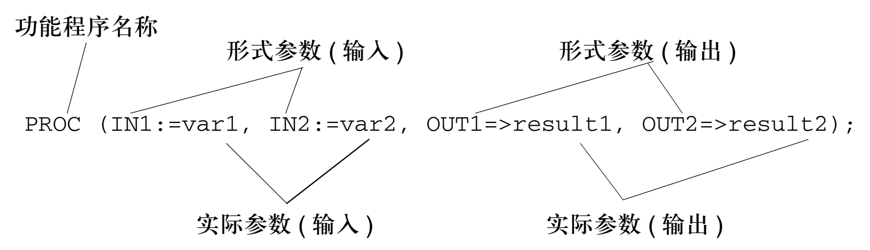
在编程中应该注意以下事项：

- 只有在输入EN=1或者不使用输入EN的情况下，才能执行功能程序(请同时参见*EN 和 ENO，469 页*)。
- 在使用 VAR_IN_OUT 变量的时候，要应用特殊的条件 (参见 *VAR_IN_OUT 变量，470 页*)。
- 有两种调用功能程序的方式：
  - 形式调用 (参见*形式调用， 467 页*) (调用具有形式参数名称的功能)
    在这种情况下，变量可以通过 => 运算符分配给输出。
  - 非形式调用 (参见*非形式调用， 468 页*) (调用没有形式参数名称的功能)

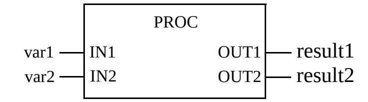
形式调用

在形式调用模式下 (通过形式参数名称进行调用)，功能程序会通过一个由功能程序名称构建的指令顺序进行调用，并带有一个括号，在括号的列表中把实际参数赋给形式参数。输入形式参数的赋值操作是通过：= 赋值的格式来完成的，输出形式参数则是通过 => 赋值的格式来完成的。输入形式参数和输出形式参数排列的顺序无关紧要。这种调用可以使用 EN 和 ENO。

调用带有实际参数名称的功能程序：



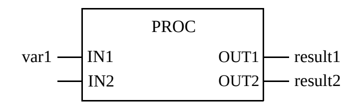
在 FBD 中调用同一个功能程序：



在形式调用中，不必为所有形式参数都赋值 (请同时参见*参数， 466 页*)。

```
PROC (IN1:=var1, OUT1=>result1, OUT2=>result2);
```

在 FBD 中调用同一个功能程序：

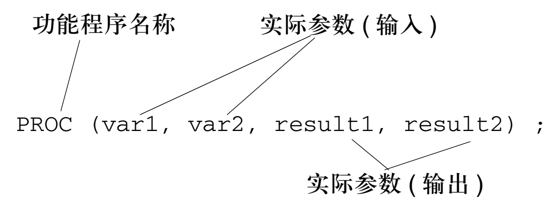


非形式调用

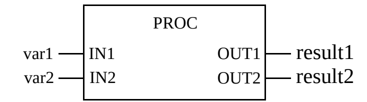
在非形式调用模式下 (不通过形式参数名称进行调用)，功能程序是通过一个由功能程序名称构建的指令顺序进行调用的，后面是一个括号，括号中有一个输入和输出实际参数的列表。实际参数的排列顺序**很重要**。实际参数列表可以紧跟在逗号之后。

这种调用**不能**使用 EN 和 ENO。

调用一个不带有形式参数名称的功能程序：



在 FBD 中调用同一个功能程序：



在非形式调用中，不必为所有形式参数都赋值 (请同时参*见参数， 466 页*)。

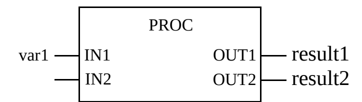
它是 IEC 61131-3 的补充内容，必须被明确激活。

如果要略过一个参数，可以使用一个空参数栏。

通过空参数栏进行调用：

```
PROC (var1, , result1, result2) ;
```

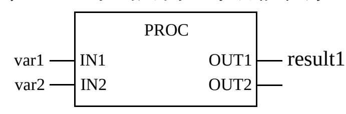
在 FBD 中调用同一个功能程序：



如果形式参数在结尾处被省略，那么可以不使用空参数栏。

```
PROC (var1, var2, result1) ;
```

在 FBD 中调用同一个功能程序：



EN 和 ENO

对所有功能程序来说，都可以对一个 EN 输入和一个 ENO 输出进行配置。

如果在调用功能程序的时候， EN 的数值等于 “0”，由功能程序所定义的算法就不会执行， ENO 会置为 “0”。

如果在调用功能程序的时候， EN 的数值等于 “1”，由功能程序所定义的算法会执行。在成功执行这些算法以后， ENO 会置为 “1”。如果在执行这些算法的时候发生了错误， ENO 会置为 “0”。

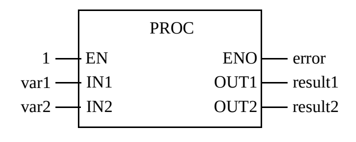
如果 ENO 被设为 “0” (因为 EN=0 或者执行过程中发生错误)，功能程序输出会置为 “0”。

功能程序的输出行为与功能在被调用的时候是否不带有 EN/ENO 或者带有 EN=1 没有关系。

如果使用了 EN/ENO，功能程序调用必须是形式调用。针对 ENO 的变量赋值操作必须通过 => 运算符来实现。

```
PROC (EN:=1, IN1:=var1, IN2:=var2,
        ENO=>error, OUT1=>result1, OUT2=>result2) ;
```

在 FBD 中调用同一个功能程序：



VAR_IN_OUT 变量

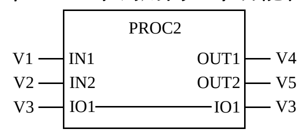
功能程序经常用来在一个输入处读入变量 (输入变量)，处理该变量，并使用同一个变量名更新输出 (输出变量)。
这种特殊类型的输入 / 输出变量也称为 VAR_IN_OUT 变量。
在使用带有 VAR_IN_OUT 变量的功能程序时，需要注意以下特殊方面。

- 所有 VAR_IN_OUT 输入都必须分配一个变量。
- VAR_IN_OUT 输入不能分配字面值或者常数。
- VAR_IN_OUT 输出**不能**赋值。
- VAR_IN_OUT 变量不能在功能块调用以外使用。

调用一个带有 VAR_IN_OUT 变量的 ST 功能块：

```
PROC2 (IN1:=V1, IN2:=V2, IO1:=V3,
       OUT1=>V4, OUT2=>V5) ;
```

在 FBD 中调用同一个功能程序：



VAR_IN_OUT 变量**不能**在功能程序调用以外使用。因而，下面的功能程序调用是**无效的**：

无效的调用，第一个例子：

InOutProc.inout := V1;	把变量 V1 赋给一个 VAR_IN_OUT 参数。 错误：不能执行操作，因为 VAR_IN_OUT 参数不能在功能块调用的外部进行访问。

无效的调用，第二个例子：

V1 := InOutProc.inout;	把一个 VAR_IN_OUT 参数赋给 V1 变量。 错误：不能执行操作，因为 VAR_IN_OUT 参数不能在功能块调用的外部进行访问。

下面的功能块调用总是有效的：

有效的调用，第一个例子：

InOutProc (inout:=V1);	调用一个带有 VAR_IN_OUT 参数的功能程序，在功能程序调用中用实际参数进行赋值。

有效的调用，第二个例子：

InOutProc (V1);	调用一个带有 VAR_IN_OUT 参数的功能程序，在功能程序调用中用实际参数进行赋值。

用户功能块 (DFB)



内容预览

本章内容

本部分包含以下内容：
- 用户功能块 (DFB)
- DFB 内部结构
- 诊断 DFB
- DFB 的类型和实例
- 使用各种语言进行实例调用

本部分内容

本部分包含以下各章：

用户功能块 (DFB) 概述

16

内容预览

本章主题

本章给出了用户功能块 (DFB) 的概述，以及实现用户功能块的各个步骤。

本章内容

本章包含以下内容：

用户功能块介绍

介绍

通过 Unity Pro 软件，您可以使用自动化语言来编制 DFB 用户功能块。 DFB 是用户为了满足应用程序的特定需要而编写的程序功能块。它包含：
- 一个或多个用梯形图 (LD)，指令表 (IL)，结构化文本 (ST) 或者功能块图 (FBD) 语言编写的代码段
- 输入 / 输出参数
- 公共或者私有内部变量

功能块可以用来对您的应用程序进行组织和优化。当您需要在应用程序中多次重复一个程序序列，或者设定一个标准的程序操作 (比如，一个用来控制马达的，考虑了本地安全性能需求的算法) 的时候，就可以使用它们。

通过对这些功能块进行相关的导出和导入操作，就可以让工作于一个或多个应用程序的编程者们共享它们。

使用 DFB 的好处

通过在应用程序中使用 DFB 功能块，您可以：
- 简化程序的设计和输入
- 增加程序的可读性
- 方便应用程序的调试 (所有由功能块操作的变量都要在它的接口进行识别)
- 减少生成的代码的数量 (与 DFB 相对应的代码只加载一次 - 虽然在程序中会多次调用，程序只生成与实例相对应的数据)

与子程序的比较

与子程序相比，使用 DFB 可以：
- 更容易地设置处理参数
- 使用针对 DFB 的内部变量，不受应用程序影响
- 可独立于应用程序对它进行测试操作

另外， LD 和 FBD 语言提供了一个 DFB 的图形视窗，方便了程序的设计和调试。

通过较早软件版本创建的 DFB

在应用程序中使用由 PL7 和 Concept 编制的 DFB 时，必须先使用产品的转换器对它们进行转换。

使用范围

下面的表格给出了 DFB 的使用范围。

功能	使用范围
可以使用 DFB 的 PLC	Premium\Atrium 和 Quantum
DFB 创建软件	Unity Pro
可以使用 DFB 的软件	Unity Pro 或者 Unity Pro Medium
创建 DFB 代码的编程语言	IL， ST， LD 或者 FBD (1)
可以使用 DFB 的编程语言	IL， ST，LD 或者 FBD (1)

(1) IL：指令表 ， ST：结构化文本， LD：梯形图， FBD：功能块图语言。

如何实现 DFB 功能块

实现过程

DFB 功能块的实现过程包含 3 个步骤：

步骤	操作
1	创建您的 DFB 模型 (称为：DFB 类型)。
2	每次在应用程序中使用 DFB 的时候，都要创建一个此功能块的副本，称为一个实例。
3	在您的应用程序中使用 DFB 实例。

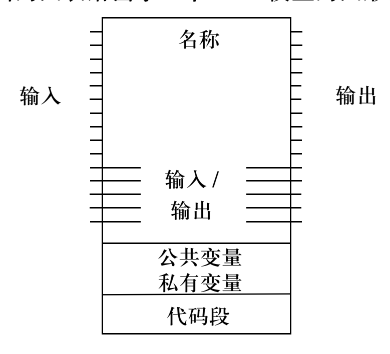
创建 DFB 类型

在这一步操作中，要设计一个用于应用程序的 DFB 模型。使用 DFB 编辑器对组成 DFB 的所有元素进行定义和编码：

- 描述功能块：名称，类型 (DFB)，激活诊断，注释。
- 功能块的结构：参数，变量，代码段。

描述 DFB 类型

下面的图表给出了一个 DFB 模型的图形演示。



功能块包含以下元素：

- 名称: DFB 类型的名称 (最多 32 个字符)。这个名称在整个库内必须是唯一的，可以在**项目设置中语言扩展**选项卡的**标识符**区域内选择扩展的字符：
- 输入：输入参数 (不包括输入 / 输出参数)。
- 输出：输出参数 (不包括输入 / 输出参数)。
- 输入 / 输出：输入 / 输出参数。
- 公共变量：可以通过应用程序访问的内部变量。
- 私有变量：嵌入于内部的变量或者 DFB，不能通过应用程序访问。
- 代码段：用 LD， IL， ST 或者 FBD 编写的 DFB 代码段。
- 注释，最多为 1024 个字符。不允许使用格式化字符 (回车， tab 键，等等)。

对于每一种 DFB，都可以通过对话框来打开一个描述文件：其中包括 DFB 的大小，参数和变量的数量，版本号，上一次修改的日期，保护级别，等等。

创建 DFB 实例

在创建了 DFB 类型以后，您就可以通过变量编辑器，或者在程序编辑器中调用该功能的时候，为该类型 DFB 定义一个实例。

使用 DFB 实例

DFB 实例可以用作

- 梯形图 (LD) 或者功能块图 (FBD) 语言中的一个标准功能块，
- 结构化文本 (ST) 或者指令表 (IL) 语言中的一个基本功能。

除了事件任务和顺序功能图 (SFC) 的转换， DFB 实例可以用在所有应用程序任务中。

存储

用户创建的 DFB 类型可以存储在功能和功能块库。

用户功能块 (DFB) 描述

17

内容预览

本章主题

本章给出了组成用户功能块的各种元素的概述。

本章内容

本章包含以下内容：

定义 DFB 内部数据

内容预览

DFB 内部数据有两种：
- 参数：输入，输出或者输入 / 输出，
- 公共或者私有变量。

DFB 的内部数据必须使用符号进行定义 (不能使用地址)。

每个参数都要定义的元素

在创建功能块的时候，每个参数都必须定义以下内容：
- 名称：DFB 类型的名称 (最多 32 个字符)。这个名称在整个库内必须是唯一的，可以在**项目设置**中**语言扩展**选项卡的**标识符**区域内选择可用的字符：
- 一个对象类型 (BOOL， INT， REAL，等等)，
- 一个注释，最多为 1024 个字符。不允许使用格式化字符 (回车，tab 键，等等)。
- 一个初始值，
- 读 / 写属性，用来定义在运行时中是否能够对变量进行写操作: R (只读) 或者 R/W (读 / 写)。只有公共变量才能定义这个属性。
- 备份属性，定义是否保存变量。

对象的类型

对于隶属于以下系列的 DFB 参数，可以定义对象的类型：
- 基本数据系列：EDT。该系列包含以下对象类型：布尔 (BOOL， EBOOL)，整数 (INT， DINT，等等)，实数 (REAL)，字符串 (STRING)，位串 (BYTE， WORD 等)，等等，
- 导出数据系列：DDT。这个系列包括数据表 (ARRAY) 和结构 (用户或者 IODDT) 对象类型，
- 泛型数据系列：ANY_INT， ANY_REAL， ANY_ARRAY，等等。比如：
  - ANY_INT 系列，其中包括整数数据 (INT， DINT， UINT， UDINT)，
  - ANY_DATE 系列，其中包括时间数据 (DATE， DATE_AND_TIME， TIME_OF_DAY)，
  - ANY_STRUCT 系列，其中包括所有的结构等。
- 功能块系列：FB。这个系列包括 EFB 和 DFB 对象类型。

各种参数可用的对象

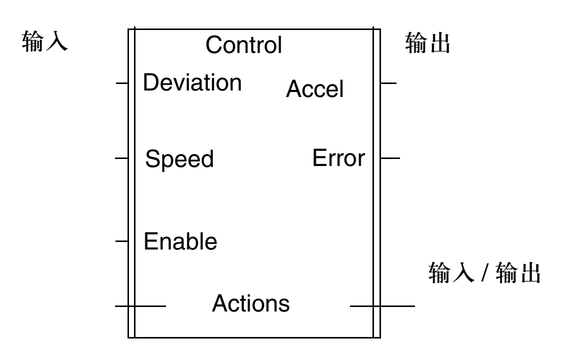
对于每一种 DFB 参数来说，有如下可用的对象系列：

对象系列	EDT	DDT [1]	IODDT	ANY_...[2]	ANY_ARRAY	FB
输入	是	是	否	是 [4]	是	否
输入 / 输出	是 [3]	是	是	是 [4]	是	否
输出	是	是	否	是 [4][5]	否	否
公共变量	是	是	否	否	否	否
私有变量	是	是	否	否	否	是
图例说明：						
(1)	导出数据系列，除了输入 / 输出导出数据类型 (IODDT)。					
(2)	所有泛型数据系列，除了 ANY_ARRAY 系列。					
(3)	除了用于 Quantum PLC 的 EBOOL 类型的静态变量。					
(4)	除了 BOOL 和 EBOOL 类型的变量。					
(5)	这些参数不能在功能块调用以外使用。					

DFB 参数

图例

下面的图例给出了 DFB 参数的例子



描述参数

下面的表格给出了每个参数的作用：

参数	最大数量	作用
输入	32 (1)	这些参数可以把应用程序的数值传递给 DFB 的内部程序。 DFB 可以在只读模式下访问它们，但是它们不能被应用程序访问。
输出	32 (2)	这些参数可以把 DFB 的数值传递给应用程序。应用程序可以在只读模式下访问它们。
输入 / 输出	32	这些参数可以把应用程序的数值传递给 DFB，经 DFB 更改后再返回给应用程序。这些参数不能被应用程序访问。

图例说明：

(1) 输入数 + 输入 / 输出数小于或等于 32

(2) 输出数 + 输入 / 输出数小于或等于 32

可以被应用程序访问的参数

唯一可以被应用程序在调用外部进行访问的参数是输出参数。如果要进行这种访问，必须在程序中使用下面的语法：**DFB_Name.Parameter_Name**

DFB_Name 表示使用的 DFB 的实例名称 (最多 32 个字符)，

Parameter_Name 表示输出参数的名称 (最多 32 个字符)。

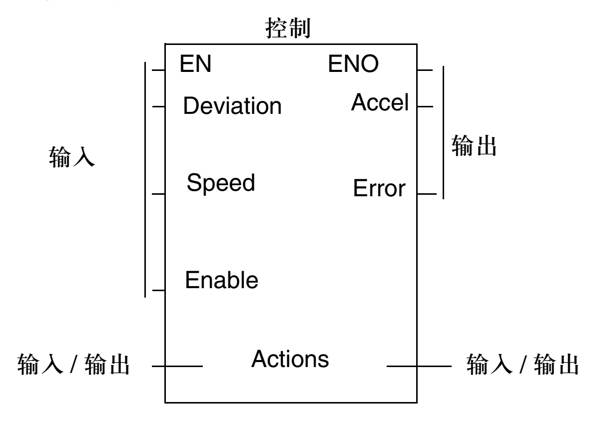
举例： `Control.Accel` 表示 DFB 实例调用的 `Control` 的输出 `Accel`

EN 和 ENO 参数

EN 是一个输入参数， **ENO** 是一个输出参数。它们都是 BOOL 类型，在 DFB 类型定义中，可以使用它们，也可以不使用它们 (可选)。

如果用户想要使用这些参数，编辑器会自动对它们进行设置：EN 是第一个输入参数， ENO 是第一个输出参数。

应用 EN\ENO 参数的例子。



如果一个实例的 EN 输入参数被赋值为 0 (FALSE)，那么：
- 构成 DFB 代码的代码段不会执行 (由系统进行管理)，
- ENO 输出参数被系统置为 0 (FALSE)。

如果一个实例的 EN 输入参数被赋值为 1 (TRUE)，那么：
- 构成 DFB 代码的代码段会执行 (由系统进行管理)，
- ENO 输出参数被系统置为 (TRUE)。

如果 DFB 例检测到了一个错误 (比如处理错误)，那么用户可以选择将 ENO 输出参数置为 0 (FALSE)。在这种情况下：
- 要么输出参数会被冻结在上一个进程的状态，直到错误消失，
- 要么用户在 DFB 代码中提供了一个功能，把输出强制为所需的状态，直到错误消失。

DFB 变量

变量描述

下面的表格给出了每种变量的作用。

变量	最大数量	作用
公共	不限	这些 DFB 内部变量可以在调整模式下被 DFB，应用程序以及用户使用。
私有	不限	这些 DFB 内部变量只能由这个功能块使用，而不能被应用程序访问。这些变量对功能块的编程来说通常是必需的，但是用户不会关心 (比如中间运算的结果，等等)。

可以被应用程序访问的变量

唯一可以被应用程序访问的变量是公共变量。如果要进行这种访问，必须在程序中使用下面的语法：**DFB_Name.Variable_Name**

DFB_Name 表示使用的 DFB 的实例名称 (最多 32 个字符)，

Variable_Name 表示公共变量的名称 (最多 8 个字符)。

例子：`Control.Gain` 表示 DFB 实例调用的 `Control` 的公共变量 `Gain`

保存公共变量

把 %S94 系统为设为 1，可以让程序或者调节器把您所修改的公共变量保存起来，从而取代其初始值 (在 DFB 实例中定义的值)。

只有在正确设置了变量的备份属性的情况下，才能进行替代操作。

DFB 代码段

常规信息

定义 DFB 进程的代码段，和声明参数的功能一样来进行。
如果设定了 IEC 选项，就可以为 DFB 创建一个单一的代码段，否则 DFB 就可以包含若干个代码段，这些代码段的数量不受限制。

编程语言

您可以使用以下语言来编写 DFB 代码段：
- 指令表 (IL)
- 结构化文本 (ST)
- 梯形图语言 (LD)
- 功能块图 (FBD)

定义代码段

代码段通过以下内容进行定义：
- 一个标识代码段的符号名称 (最多 32 个字符)
- 一个定义代码段执行的生效条件
- 一个注释 (最多 256 个字符)
- 一个保护属性 (无保护，写保护代码段，读 / 写保护代码段)

编程规则

在执行 DFB 代码段的时候，只能使用您为功能块定义的参数 (输入，输出和输入 / 输出参数以及内部变量)。所以，除了系统字和系统位 (%Si， %SWi 和 %SDi) 以外， DFB 功能块不能使用应用程序的全局变量，也不能使用输入 / 输出对象。
DFB 对它的参数有最大访问权限 (读和写)。

代码的例子

下面的程序给出了一个结构化文本代码的例子

```
CHR_200:=CHR_100;
CHR_114:=CHR_104;
CHR_116:=CHR_106;
RESET DEMARRE;
(*On incremente 80 fois CHR_100*)
FOR CHR_102:=1 TO 80 DO
          INC CHR_100;
          WHILE((CHR_104-CHR_114)<100)DO
          IF(CHR_104>400) THEN
EXIT;
          END_IF;
          INC CHR_104;
          REPEAT
          IF(CHR_106>300) THEN
EXIT;
          END_IF;
          INC CHR_106;
          UNTIL ((CHR_100-CHR_116)>100)
          END_REPEAT;
          END_WHILE;
          (* On boucle tant que CHR_106)
          IF (CHR_106=CHR_116)
          THEN EXIT;
          ELSE
              CHR_114:=CHR_104;
              CHR_116:=CHR_106;
          END_IF;
          INC CHR_200;
END_FOR;
```

用户功能块 (DFB) 实例

18

内容预览

本章主题

本章给出了创建和执行 DFB 实例的概述

本章内容

本章包含以下内容：

创建 DFB 实例

DFB 实例

DFB 实例是模型 (DFB 类型) 的一个副本：
- 它使用 DFB 类型代码 (代码没有复制)，
- 它创建一个实例专用的数据区，这个实例是 DFB 类型参数和变量的副本。该区域位于应用程序的数据区。

您必须用一个名称来标识您所创建的每个 DFB 实例，可以在**项目设置中语言扩展**选项卡的**标识符**区域内选择扩展字符。第一个字符必须是一个字母！不能使用关键字和符号。

创建实例

您可以根据需要，对一个 DFB 类型可创建无限多个实例；限制实例数量的唯一因素是 PLC 内存的大小。

初始值

您在创建 DFB 类型的过程中所定义的参数和公共变量，其初始值可以针对每个 DFB 实例进行修改。
并不是所有的 DFB 参数都有初始值。

修改 DFB 实例中的元素的初始值

	EDT (除了字符串类型)	**字符串类型**	**EDT**	**DDT 结构**	**FB**	**ANY_ARRAY**	**IODDT**	**ANY_...**
输入	是	否	否	否	-	否	-	否
输入 / 输出	否	否	否	否	-	否	否	否
输出	是	是	否	是	-	-	-	否
公共变量	是	是	是	是	-	-	-	-
私有变量	否	否	否	否	否	-	-	-

修改 DFB 类型中的元素的初始值

	EDT (除了字符串类型)	**字符串类型**	**EDT**	**DDT 结构**	**FB**	**ANY_ARRAY**	**IODDT**	**ANY_...**
输入	是	否	否	否	-	否	-	否
输入 / 输出	否	否	否	否	-	否	否	否
输出	是	是	否	是	-	-	-	否
公共变量	是	是	是	是	-	-	-	-
私有变量	是	是	是	是	否	-	-	-

执行 DFB 实例

操作

执行 DFB 实例的步骤如下所示：

步骤	操作
1	为输入和输入 / 输出参数加载数值。在初始化阶段 (或者在冷重启阶段)，所有未赋值的输入都会使用在 DFB 类型中定义的初始值。在接下来的操作中，它们会保留最后一次的赋值。
2	执行 DFB 的内部程序。
3	写输出参数。

调试 DFB

Unity Pro 软件提供了若干种 DFB 调试工具：

- 动态数据表：所有参数，公共和私有变量都会以实时的方式显示和动化。对象可以进行修改和强制操作。
- 断点，逐步操作和程序诊断
- 操作员界面：用于统一调试

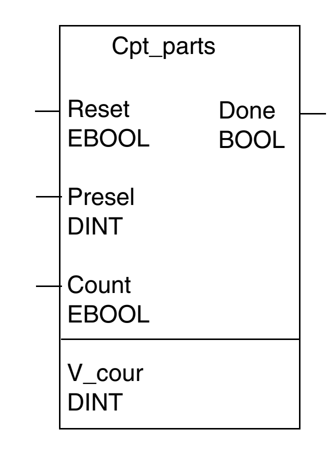
导出功能块 (DFB) 编程的例子

常规信息

下面给出了一个使用 DFB 编写计数器的例子，以便于用户了解相关内容。

DFB 类型的特性

用来创建计数器的 DFB 类型如下所示：



DFB 类型 `Cpt_parts` 的元素如下所示：

元素	描述
DFB 类型的名称	**Cpt_parts**
输入参数	• **Reset**：重置计数器 (EBOOL 类型) • **Presel**：预先设定计数器的数值 (DINT 类型) • **Count**：向上计数器的输入 (EBOOL 类型)
输出参数	**Done**：预先设定的数值达到了输出值 (BOOL 类型)
公共内部变量	**V_cour**：计数器的当前数值 (DINT 类型)

计数器的操作

计数器必须按照如下方式进行操作：

阶段	描述
1	DFB 在 **Count** 输入的上升沿计数。
2	它所计数的上升沿的数量会存储在变量 `V_cour` 中。 `Reset` 输入的上升沿可以复位该变量。
3	当计数的上升沿的数量等于预先设定的数值时，`Done` 输出会置为 1。`Reset` 输入的上升沿可以复位该变量。

DFB 的内部程序

DFB 类型 `Cpt_parts` 的内部程序通过结构化文本以如下方式定义：

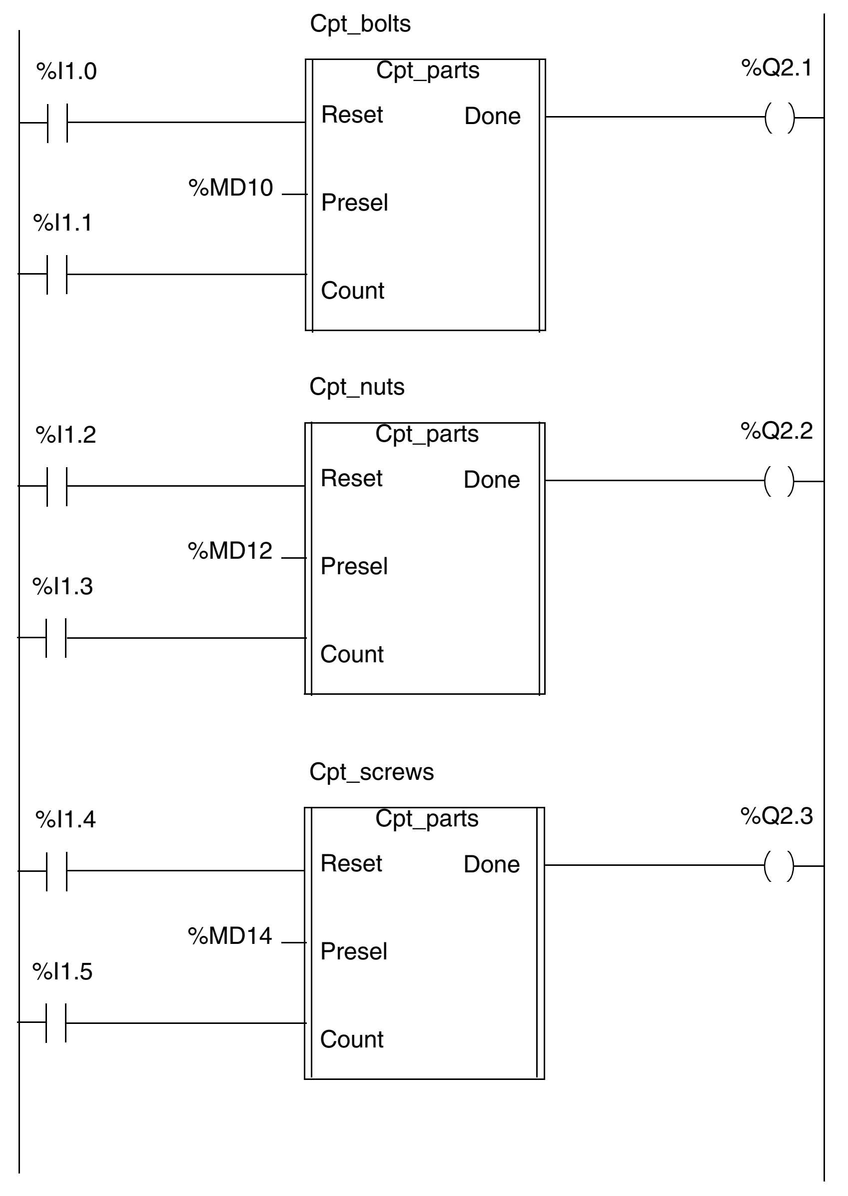
```
!(*Programming of the Cpt_parts DFB*)
IF RE (Reset) THEN
        V_cour:=0;
END_IF;
IF RE (Count) THEN
        V_cour:=V_cour+1;
END_IF;
IF(V_cour>=Presel) THEN
        SET (Done);
ELSE
        RESET (Done);
END_IF;
```

应用的例子

让我们假定您的应用程序需要对三个部件类型进行计数 (比如螺栓，螺母和螺杆)。可以使用 DFB 类型 `Cpt_parts`3 次 (3 个实例) 来完成这些计数工作。

每种部件要达到的数量分别在字 %MD10， %MD12 和 %MD14 中进行定义。在达到了部件的数量以后，计数器会向输出发出一个命令 (%Q1.2.1， %Q1.2.2 或 %Q1.2.3)，停止相关部件的计数操作。

下面的应用程序以梯形图语言输入。3 个 DFB (实例) Cpt_bolts， Cpt_nuts 和 Cpt_screws 用来对不同的部件进行计数。



在各种编程语言中使用 DFB

19

内容预览

本章主题

本章给出了用各种编程语言进行 DFB 实调用的概述。

本章内容

本章包含以下内容：

在程序中使用 DFB 的规则

常规信息

除了事件任务和 SFC 程序的转换以外，在所有语言 [指令表 (IL)，结构化文本 (ST)，梯形图 (LD) 和功能块图 (FBD)] 以及应用程序的所有任务 (代码段，子程序，等等) 中都可以使用 DFB 实例。

一般使用规则

在使用 DFB 的时候，不管用的是什么语言，您都必须遵守如下规则：

- 不必连接所有输入，输入 / 输出或者输出参数，但是下面的参数一定要赋值：
  - 泛型数据类型的输入参数 (ANY_INT， ANY_ARRAY，等等)，
  - 输入 / 输出参数，
  - 泛型数据类型的输出参数 (除了数据表)(ANY_INT， ANY_REAL，等等)。
- 未连接的输入参数会保持先前调用的数值，如果相关的功能块从未被调用过，那么它们会保持定义的初始化数值。
- 所有分配给输入，输入/输出和输出参数的对象都必须与创建DFB类型时所定义的类型相同 (比如：如果输入参数 “speed”定义了 INT 类型，那么您就不能把它赋给 DINT 或 REAL 类型的数据)。
  唯一的例外是用于输入和输出参数 (不是输入 / 输出参数) 的 BOOL 和 EBOOL 类型，它们可以混用。
  例子：输入参数 “Validation”可以定义为 BOOL 类型，并与一个 EBOOL 类型的内部字 %Mi 相关联。不过，在 DFB 类型的内部代码中，输入参数实际上有 BOOL 类型的属性 (它不能进行边沿处理)。

参数赋值

下面的表格对在各种编程语言中进行参数赋值的可能性进行了总结：

参数	类型	参数赋值 (1)	赋值
输入	EDT (2)	相连的，数值，对象或者表达式	可选
	BOOL	相连的，数值，对象或者表达式	可选
	DDT	相连的，数值或者对象	可选
	ANY_...	相连的或者对象	强制
	ANY_ARRAY	相连的或者对象	强制
输入 / 输出	EDT	相连的或者对象	强制
	DDT	相连的或者对象	强制
	IODDT	相连的或者对象	强制
	ANY_...	相连的或者对象	强制
	ANY_ARRAY	相连的或者对象	强制
Outputs	EDT	相连的或者对象	可选
	DDT	相连的或者对象	可选
	ANY_...	相连的或者对象	强制
	ANY_ARRAY	相连的或者对象	可选

(1) 在梯形图 (LD) 或者功能块图 (FBD) 语言中的 “相连的”。在指令表 (IL) 或者结构化文本 (ST) 语言中的数值或对象。
(2) 除了 BOOL 类型的参数

在 DFB 中使用 IODDT

内容预览

下面的表格给出了可以用于 Premium 和 Quantum (参见*模拟量应用程序， 499 页)* PLC 的 DFB 的各种 IODDT。

离散量应用程序

下面的表格给出了 Premium PLC 离散量应用程序所用到的 IODDT。

IODDT 系列	是否可以用于 DFB	
T_DIS_IN_GEN	-	否
T_DIS_IN_STD	-	否
T_DIS_EVT	-	否
T_DIS_OUT_GEN	-	否
T_DIS_OUT_STD	-	否
T_DIS_OUT_REFLEX	-	否

模拟量应用程序

下面的表格给出了 Premium PLC 模拟量应用程序所用到的 IODDT。

IODDT 系列	是否可以用于 DFB	
T_ANA_IN_GEN	-	否
T_ANA_IN_STD	-	否
T_ANA_IN_CTRL	是	-
T_ANA_IN_EVT	是	-
T_ANA_OUT_GEN	-	否
T_ANA_OUT_STD	-	否

计数应用程序

下面的表格给出了 Premium PLC 计数应用程序所用到的 IODDT。

IODDT 系列	是否可以用于 DFB	
T_COUNT_ACQ	是	-
T_COUNT_HIGH_SPEED	是	-
T_COUNT_STD	是	-

电子凸轮应用程序

下面的表格给出了 Premium PLC 电子凸轮应用程序所用到的 IODDT。

IODDT 系列	是否可以用于 DFB	
T_CCY_GROUP0	-	否
T_CCY_GROUP1_2_3	-	否

轴控应用程序

下面的表格给出了 Premium PLC 轴控应用程序所用到的 IODDT。

IODDT 系列	是否可以用于 DFB	
T_AXIS_AUTO	是	-
T_AXIS_STD	是	-
T_INTERPO_STD	是	-
T_STEPPER_STD	是	-

Sercos 应用程序

下面的表格给出了 Sercos 应用程序用到的 IODDT。

IODDT 系列	是否可以用于 DFB	
T_CSY_CMD	是	-
T_CSY_TRF	是	-
T_CSY_RING	是	-
T_CSY_IND	是	-
T_CSY_FOLLOW	是	-
T_CSY_COORD	是	-
T_CSY_CAM	是	-

通信应用程序

下面的表格给出了 Premium PLC 通信应用程序所用到的 IODDT。

IODDT 系列	是否可以用于 DFB	
T_COM_STS_GEN	是	-
T_COM_UTW_M	是	-
T_COM_UTW_S	是	-
T_COM_MB	是	-
T_COM_CHAR	是	-
T_COM_FPW	是	-
T_COM_MBP	是	-
T_COM_JNET	是	-
T_COM_ASI	是	-
T_COM_ETY_1X0	是	-
T_COM_ETY_210	是	-
T_COM_IBS_128	是	-
T_COM_IBS_242	是	-
T_COM_PBY	是	-
T_COM_CPP100	是	-
T_COM_ETYX103	是	-
T_COM_ETHCOPRO	是	-

调整应用程序

下面的表格给出了调整应用程序所用到的 IODDT。

IODDT 系列	是否可以用于 DFB	
T_PROC_PLOOP	是	-
T_PROC_3SING_LOOP	是	-
T_PROC_CASC_LOOP	是	-
T_PROC_SPP	是	-
T_PROC_CONST_LOOP	是	-

称重应用程序

下面的表格给出了 Premium PLC 称重应用程序所用到的 IODDT。

IODDT 系列	是否可以用于 DFB	
T_WEIGHING_ISPY101	是	-

所有应用程序公用

下面的表格给出了 Premium PLC 所有应用程序都能用到的 IODDT。

IODDT 系列	是否可以用于 DFB	
T_GEN_MOD	-	否

模拟量应用程序

下面的表格给出了 Quantum PLC 模拟量应用程序所用到的 IODDT。

IODDT 系列	是否可以用于 DFB	
T_ANA_IN_VE	-	否
T_ANA_IN_VWE	-	否
T_ANA_BI_VWE	-	否
T_ANA_BI_IN_VWE	-	否

计数应用程序

下面的表格给出了 Quantum PLC 计数应用程序所用到的 IODDT。

IODDT 系列	是否可以用于 DFB	
T_CNT_105	-	否

在梯形图语言的程序中使用 DFB

原则

在梯形图语言中有两种调用 DFB 功能块的方式：
- 在一个操作功能块内进行文本调用，该功能块中参数的语法和约束条件与结构化文本语言一致。
- 通过图形调用

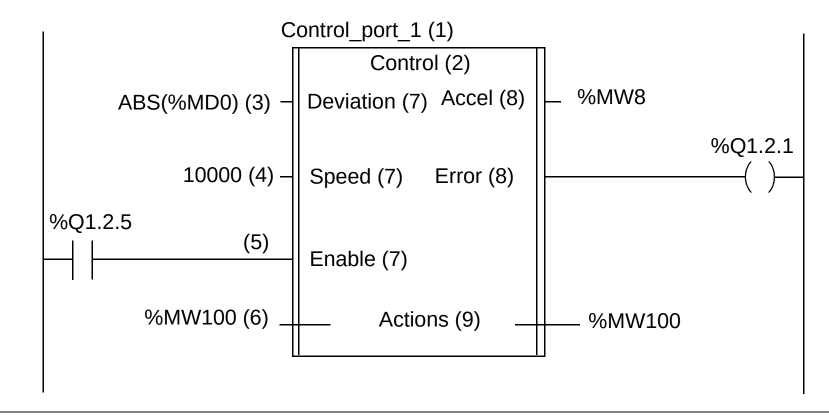
功能块的输入可以被连到或者分配一个数值，对象或者表达式。不论在哪种情况下，外部元素的类型 (数值，表达式的运算结果，等等) 都必须与输入参数一致。

DFB 功能块必须至少有一个与之相连的布尔输入和一个输出 (如果需要的话)。您可以使用 EN 输入参数和 ENO 输出参数来实现 (参见后文关于这些参数的描述)。

对于 ANY_ARRAY 类型的输入，泛型数据类型的输出 (ANY_...) 以及 DFB 功能块的输入 / 输出，必须对它们进行连接或者赋值操作。

DFB 功能块的图形演示

下面的图例给出了 DFB 编程的一个简单的例子：



DFB 功能块的元素

下面的表格给出了上图所标出的 DFB 功能块的各个元素：

标号	元素
1	DFB (实例) 名称
2	DFB 类型的名称
3	通过一个表达式进行赋值的输入
4	通过一个数值进行赋值的输入
5	连接的输入
6	通过一个对象 (地址或符号) 进行赋值的输入
7	输入参数
8	输出参数
9	输入 / 输出参数

使用 EN\ENO 参数

参见 *EN 和 ENO 参数， 483 页*

在结构化文本语言的程序中使用 DFB

原则

在结构化文本中，用户功能块的调用通过一个 DFB 调用来实现：DFB 实例的名称后面带有一个参数列表。参数会在括号中列出，彼此之间用逗号分隔。
DFB 调用可以是以下两种类型之一：

- 一个形式调用，其中参数是赋值 (参数 = 数值)。在这种情况下，参数在列表中输入的顺序无关紧要。
  EN 输入参数和 ENO 输出参数可以用来控制功能块的执行，
- 一个非形式调用，其中参数是数值 (表达式，对象或者立即值)。在这种情况下，参数在列表中输入的顺序必须符合 DFB 输入参数的顺序，包括未赋值的输入 (参数是一个空栏)。不能使用 EN 和 ENO 参数。

```
DFB_Name (argument 1,argument 2,....,argument n)
```

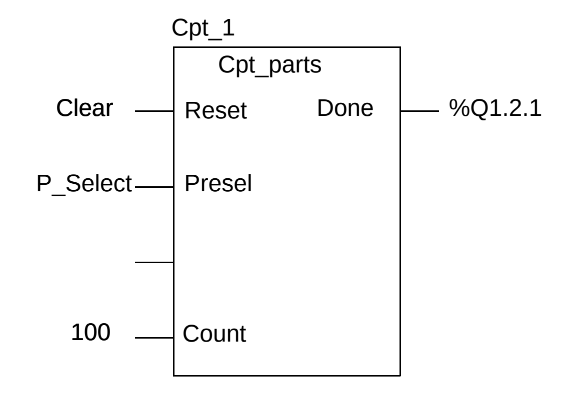
注意：ANY_ARRAY 类型的输入，泛型数据类型的输出 (ANY_...) 以及 DFB 的输入 / 输出必须赋值。

使用 EN\ENO 参数

参见 *EN 和 ENO 参数， 483 页*

DFB 的例子

下面是一个简单的例子，它解释了结构化文本语言中的各种 DFB 调用。它是 `Cpt_parts`：类型 DFB 的 `Cpt_1` 实例



形式 DFB 调用

形式 DFB 调用 `Cpt_1` 通过下面的语法来实现：

```
Cpt_1 (Reset:=Clear, Presel:=P_Select, Count:=100,
Done=>%Q1.2.1);
```

其中通过一个数值 (表达式，对象或者立即值参数) 进行赋值的输入参数在参数列表中输入，它的语法如下：

```
Cpt_1 (Reset:=Clear, Presel:=P_Select, Count:=100);
...
%Q1.2.1:=Cpt_1.Done;
```

顺序的元素

下面的表格给出了进行形式 DFB 调用时所用到的程序顺序的各个元素：

元素	含义
Cpt_1	DFB 实例名称
Reset, Presel, Count	输入参数
:=	一个输入的赋值符号
Clear	一个输入的赋值对象 (符号)
100	一个输入的赋值数值
Done	输出参数
=>	一个输出的赋值符号
%Q1.2.1	一个输出的赋值对象 (地址)
;	顺序结束符号
,	参数分隔符号

非形式 DFB 调用

非形式 DFB 调用 `Cpt_1` 通过下面的语法来实现：

```
Cpt_1 (Clear, %MD10, , 100);
...
%Q1.2.1:=Cpt_1.Done;
```

顺序的元素

下面的表格给出了进行非形式 DFB 调用时所用到的程序顺序的各个元素：

元素	含义
Cpt_1	DFB 实例名称
Clear, %MD10, ,100	输入的赋值对象或数值。未赋值的输入以一个空栏表示
;	顺序结束符号
,	参数分隔符号

在指令表的程序中使用 DFB

原则

在指令表中，用户功能块通过 CAL 指令调用，该指令后面带有用作操作数的 DFB 实例名称，以及一个参数列表 (可选)。参数列表置于括号中，参数间以逗号分隔。

在指令表中，有三种调用 DFB 的方式：

- 指令 CAL DFB_Name 后面带有一个赋值的参数列表 (参数 = 数值)。在这种情况下，参数在列表中输入的顺序无关紧要。
  EN 输入参数可以用来控制功能块的执行，
- 指令 CAL DFB_Name 后面带有一个参数列表，参数为数值 (表达式，对象或者立即值)。在这种情况下，参数在列表中输入的顺序必须符合 DFB 输入参数的顺序，其中包括未赋值的输入 (参数是一个空栏)。
  不能使用 EN 和 ENO 参数，
- 指令 CAL DFB_Name 后面不带有参数列表。在这种情况下，在这个指令之前必须通过一个寄存器对输入参数进行赋值：首先把数值载入 (加载)，然后赋给输入参数 (存储)。参数 (LD/ST) 赋值的顺序无关紧要；不过，在执行 CAL 命令之前，您必须对所有需要的输入参数进行赋值。不能使用 EN 和 ENO 参数。

```
CAL DFB_Name (argument 1,argument 2,...,argument n)
```

或者

```
LD Value 1
ST Parameter 1
...
LD Value n
ST Parameter n
CAL DFB_Name
```

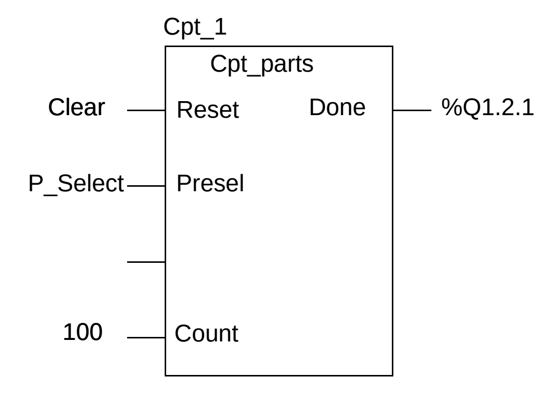
注意：ANY_ARRAY 类型的输入，泛型数据类型的输出 (ANY_...) 以及 DFB 的输入 / 输出必须赋值。

使用 EN\ENO 参数

参见 *EN 和 ENO 参数， 483 页*。

DFB 的例子

下面的例子解释了在指令表中所使用的各种 DFB 调用。它是 `Cpt_parts`：类型 DFB 的 `Cpt_1` 实例



当参数为赋值时的 DFB 调用

当参数为赋值时， DFB 调用 `Cpt_1` 通过以下语法来实现：

```
CAL Cpt_1 (Reset:=Clear, Presel:=%MD10, Count:=100,
Done=>%Q1.2.1)
```

其中通过一个数值 (表达式，对象或者立即值) 进行赋值的输入参数在参数列表中输入，它的语法是：

```
CAL Cpt_1 (Reset:=Clear, Presel:=%MD10, Count:=100)
...
LD Cpt_1.Done
ST %Q1.2.1
```

为了增加应用程序的可读性，您可以在分隔参数的逗号后面输入回车，这样程序顺序就会具有如下所示的语法：

```
CAL Cpt_1(
Reset:=Clear,
Presel:=%MD10,
Count:=100,
Done=>%Q1.2.1)
```

DFB 调用程序的元素

下面的表格给出了 DFB 调用程序的各种元素：

元素	含义
CAL	DFB 调用指令
Cpt_1	DFB 实例名称
Reset, Presel, Count	输入参数
:=	一个输入的赋值符号
Clear, %MD10, 100	输入的赋值对象或数值
Done	输出参数
=>	一个输出的赋值符号
%Q1.2.1	一个输出的赋值对象
,	参数分隔符号

当参数为数值时的 DFB 调用

当参数为数值时， call 调用 `Cpt_1` 通过以下语法来实现：

```
CAL Cpt_1 (Clear, %MD10,, 100)
...
LD Cpt_1.Done
ST %Q1.2.1
```

DFB 调用程序的元素

下面的表格给出了 DFB 调用程序的各种元素：

元素	含义
CAL	DFB 调用指令
Cpt_1	DFB 实例名称
Clear, %MD10, 100	输入的赋值对象或数值
,	参数分隔符号

没有参数的 DFB 调用

如果没有参数， DFB 调用 Cpt_1 通过以下语法来实现：

```
LD Clear
ST Cpt_1.Reset
LD %MD10
ST Cpt_1.Presel
LD 100
ST Cpt_1.Count
CAL Cpt_1(
...
LD Cpt_1.Done
ST %Q1.2.1
```

DFB 调用程序的元素

下面的表格给出了 DFB 调用程序的各种元素：

元素	含义
LD Clear	用来把 Clear 数值载入到寄存器的加载指令
ST Cpt_1.Reset	把寄存器内容赋给输入参数 Cpt_1.Reset 的赋值指令
CAL Cpt_1(	用于 DFBCpt_1 的调用指令

在功能块图语言的程序中使用 DFB

原则

在 FBD (功能块图) 语言中，用户功能块和梯形图语言中的表示方式相同，它以图形的方式被调用。

用户功能块的输入可以通过一个数值，立即值对象或者表达式进行连接和赋值。无论是哪种情况，外部元素的类型都必须与输入参数的类型一致。

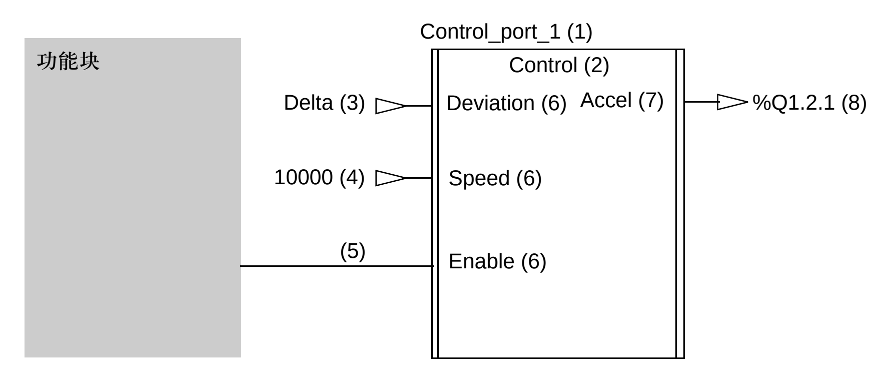
DFB 的一个输入只能分配一个对象 (链接到具有同样变量的另外一个功能块)。不过，一个输出可以与多个对象相连接。

DFB 功能块必须至少有一个与之相连的布尔输入和一个输出 (如果需要的话)。您可以使用 EN 输入参数和 ENO 输出参数来实现。

对于 ANY_ARRAY 类型的输入，泛型数据类型的输出 (ANY_...) 以及 DFB 功能块的输入 / 输出，必须对它们进行连接或者赋值操作。

DFB 功能块的图形演示

下面的图例给出了 DFB 编程的一个简单的例子。



DFB 功能块的元素

下面的表格给出了上图所标出的 DFB 功能块的各个元素：

标号	元素
1	DFB（实例）名称
2	DFB 类型的名称
3	通过一个对象（符号）进行赋值的输入
4	通过一个数值进行赋值的输入
5	相连的输入
6	输入参数
7	输出参数
8	通过一个对象（地址）进行赋值的输入

使用 EN\ENO 参数

参见 EN 和 ENO 参数，483 页。

用户诊断 DFB

20

内容预览

本章主题

本章描述了如何在 Premium， Atrium 和 Quantum PLC 上创建和使用用户诊断功能块。

本章内容

本章包含以下内容：

用户诊断 DFB 的介绍

常规信息

通过 Unity Pro 应用程序，您可以创建自己的诊断 DFB。

这些诊断 DFB 是您事先通过**诊断属性**配置的标准 DFB，您可以在其中使用下面两个功能：

- REGDFB，用来保存警报日期。
- DEREG，用来重新登记警报。

您可以通过这些 DFB 来监视进程。它们会在观测器中自动报告您所选择的信息。这样一来，您就可以监视状态的变化或者进程的更改。

好处

该功能主要有以下好处：

- 诊断已经集成在项目中，因而可以在开发阶段进行构思，从而更好地满足用户需要。
- 错误的日期确定和记录系统是在错误源完成的 (在 PLC 中)，这意味着相关的信息能够准确反映进程的状态。
- 您可以连接多个观测器 (Unity Pro，Magelis，Factory Cast)，它们会把进程的准确状态传送给用户。每一个观测器都是独立的，与此同时，在任何一个观测器上所进行的操作 (比如一个确认) 都会在其他观测器上自动显示出来。

如何创建用户诊断 DFB

内容预览

您可以通过如下两种方式之一来创建诊断 DFB。

- 您可以创建一个标准的 DFB，然后定义它的数值并对它进行编程，这样一来，它就是一个诊断 DFB。
- 您可以使用我们所提供的 USER_DIAG_ST_MODEL 模型，或者自己已经拥有的模型。

下面给出了这两种方式的介绍。

注意：如果要了解用户诊断 DFB 的工作机制，请参考在诊断库中所描述的 USER_DIAG_ST_MODEL 模型。

在不使用模型的情况下创建诊断 DFB

下面的表格给出了在不使用预定义模型的情况下创建诊断 DFB 的流程。

步骤	操作
1	在项目浏览器的结构视图中，用右键点击**导出 DFB 类型**目录。 **结果**：一个快捷菜单被打开。
2	启动**打开**命令。 **结果**：数据编辑器窗口被打开。该窗口位于 **DFB 类型**选项卡的正下方，在窗口中给出项目的 DFB 列表。
3	在**名称**栏内 (用一个箭头表示) 选择第一个空单元，输入新 DFB 类型的**名称**，然后通过**输入**进行确认。 **结果**：创建了 DFB 类型的空结构。这个新的 DFB 会被添加到现有 DFB 的列表中去。它同时也会在**导出 DFB 类型**目录树下显示出来。
4	在数据编辑器中，用右键点击你刚刚创建的 DFB 类型。 **结果**：出现上下文菜单。
5	选择**属性**命令。 **结果**：出现数据属性窗口。
6	点击**诊断**按钮。 **结果**：复选框被选中，以红色表示出来。这类 DFB 现在就成为诊断 DFB。如果在项目设置中要求了应用程序诊断，那么在生成项目的时候，会在 PLC 中包含项目诊断服务。
7	创建输入、输出、输入 / 输出以及 DFB 的内部变量。 **注意**：在输入的属性中，记住要为被 DFB 监视的输入选择相应的**诊断**选项。

步骤	操作
8	通过 REGDFB 功能来写 DFB 的代码，以便于保存警报并确定警报日期，通过 DEREG 功能来重新登记警报。 **注意：**如果你不了解这种 DFB，请参看 USER_DIAG_ST_MODEL 用户诊断 DFB 模型的代码描述。
9	运行一个项目分析，以便对新的 DFB 类型进行验证。 **结果：**您创建了新的 DFB 类型。您现在可以在程序中创建一个该类 DFB 的实例并插入到程序中。当您的 PLC 处于运行模式时，观测器会根据您编写的操作来显示警报。

通过 USER_DIAG_ST_MODEL 模型来创建诊断 DFB

下面的表格给出了在使用预定义 USER_DIAG_ST_MODEL 模型的情况下创建诊断 DFB 的流程。

步骤	操作
1	在项目浏览器的结构视图中，用右键点击**导出 DFB 类型**目录。 **结果：**一个快捷菜单被打开。
2	选择**从库中获取**命令。 **结果：**出现**库浏览器**窗口。
3	在诊断库中选择 USER_DIAG_ST_MODEL 诊断 DFB 模型，然后点击 >> 按钮把它发送给您的项目。
4	点击**确认**对传送进行确认。 **结果：**该模型出现在项目浏览器中。
5	用您想创建的用户 DFB 类型名称来代替 USER_DIAG_ST_MODEL 的名称。
6	修改和创建 DFB 的输入，输出，输入 / 输出和内部变量。 **注意：**在输入的属性中，记住要为被 DFB 监视的输入选择相应的**诊断**选项。
7	用该模型来写 DFB 的代码。 **注意：**如果你不了解这种 DFB，请参看用户诊断 DFB 模型的代码描述。
8	运行一个项目分析，以便对新的 DFB 类型进行验证。 **结果：**您创建了新的 DFB 类型。您现在可以在程序中创建一个该类 DFB 的实例并插入到程序中。当您的 PLC 处于运行模式时，观测器会根据您编写的操作来显示警报。 **注意：**观测器所显示的消息是与 DFB 实例相关的注释。所以您必须要为创建的每一个实例填写该注释。

附录



介绍

内容预览

本附录提供了与本文相关的附加信息。

本附录内容

本附录包含以下各章：

遵循的 IEC 标准



内容预览

概述

本章给出了 IEC 61131-3 的遵循性表格：

本章内容

本章包含以下各节：

A.1 关于 IEC 61131-3 的常规信息

关于 IEC 61131-3 遵循性的常规信息

内容预览

IEC 61131-3 标准 (及其子条款 1.4) 对用于可编程控制器的一套标准化编程语言的语法和语义进行了详细说明。这些编程语言包括两种文本语言：IL (指令表) 和 ST (结构化文本)，以及两种图形语言：LD (梯形图) 和 FBD (功能块图)。它还定义了顺序功能图 (SFC) 语言的元素，它们用来构建可编程控制器程序和功能块的内部组织。除此以外，还定义了用来支持把可编程控制器程序安装到可编程控制器系统中去的配置元素。

注意：Unity Pro 的编程语言采用英文缩写。

另外，还定义了旨在为不同可编程控制器以及其他自动化系统组件之间进行通信提供方便的功能。

Unity Pro 遵循的 IEC 61131-3 标准

当前版本的 Unity Pro 编程系统支持在**标准**中所定义的一个遵循性语言元素子集。

此处的遵循性表示：
- **标准**允许实现 IEC 编程系统的用户在**功能表**中选择或者摒弃某些特定的语言特性，甚至是整个语言，该**功能表**是隶属于技术规范的内容。一个系统如果要符合**标准**，就必须根据在**标准**中所给出的技术规范来选择和实现相应的特性。
- 另外，**标准**允许实现 IEC 编程系统的用户在一个交互式编程环境中使用业已定义的编程语言元素。根据**标准**所给出的明确说明，此种环境的技术规范不在标准所涉及的范围之内，因而用户拥有一定的自由度，可以针对特定的语言元素来提供优化的表达和管理措施。
- 通过这种自由度，Unity Pro 可以为同时管理 IEC 语言元素“配置”和“资源”而引入 “项目”的概念。在管理变量声明或者功能块实例化的时候，它也会用到这种自由度。

IEC 标准表格

下面的遵循性声明和其后的表格阐明了**标准**所支持的功能以及其他专门的应用信息。

A.2 IEC 遵循性表格

内容预览

概述

这个系统满足 IEC 61131-3 在下面的表格中就语言和功能所提出的要求。

本节内容

本节包含以下主题：

公共元素

公共元素

公共元素的 IEC 遵循性表格：

表格号码	特性号码	特性描述
1	2	小写字母
	3a	数字符 (#)
	4a	美元符号 ($)
	5a	竖线 (|)
2	1	大写字母和数字
	2	大写和小写字母，数字，内部下划线
	3	大写和小写字母，数字，开始下划线或者内部下划线
3	1	注释
3a	1	程序
4	1	整数字面值
	2	实数字面值
	3	带有指数的实数字面值
	4	基数为 2 的字面值
	5	基数为 8 的字面值
	6	基数为 16 的字面值
	7	布尔 0 和 1
	8	布尔 FALSE 和 TRUE
	9	型态字面值
5	1	单字节字符串
	3	单字节型态字符串字面值
6	2	美元符号
	3	单引号
	4	换行
	5	新行
	6	换页 (页码)
	7	回车
	8	Tab 键
	9	双引号

表格号码	特性号码	特性描述
7	1a	没有下划线的持续时间字面值：短前缀
	1b	长前缀
	2a	带有下划线的持续时间字面值：短前缀
	2b	长前缀
8	1	日期字面值 (长前缀)
	2	日期字面值 (短前缀)
	3	日时间字面值 (长前缀)
	4	日时间字面值 (短前缀)
	5	日期和时间字面值 (长前缀)
	5	日期和时间字面值 (短前缀)
10	1	BOOL 数据类型
	3	INT 数据类型
	4	DINT 数据类型
	7	UINT 数据类型
	8	UDINT 数据类型
	10	REAL 数据类型
	12	TIME 数据类型
	13	DATE 数据类型
	14	TIME_OF_DAY 或 TOD 数据类型
	15	DATE_AND_TIME 或 DT 数据类型
	16	STRING 数据类型
	17	BYTE 数据类型
	18	WORD 数据类型
	19	DWORD 数据类型
12	4	数组数据类型
	5	结构化数据类型
14	4	数组数据类型的初始化
	6	导出结构化数据类型的初始化

表格号码	特性号码	特性描述
15	1	输入位置
	2	输出位置
	3	内存位置
	4	单一位的大小 (X 前缀)
	5	单一位的大小 (没有前缀)
	7	字 (16 位) 的大小
	8	双字 (32 位) 的大小
	9	长 (四倍长) 字 (64 位) 的大小
17	3	符号变量位置的声明 (*注释 5， 529 页*)
	4	数组位置赋值 (*注释 5， 529 页*)
	5	符号变量的自动内存分配
	6	数组声明 (*注释 11， 531 页*)
	7	保持性数组声明 (*注释 11， 531 页*)
	8	结构化变量的声明
18	1	直接表示的变量的初始化 (*注释 11， 531 页*)
	3	符号变量的位置和初始值赋值
	4	数组位置赋值和初始化
	5	符号变量的初始化
	6	数组初始化 (*注释 11， 531 页*)
	7	保持性数组声明和初始化 (*注释 11， 531 页*)
	8	结构化变量的初始化
	9	常数的初始化
	10	功能块实例的初始化
19	1	否定输入
	2	否定输出
19a	1	形式功能 / 功能块调用
	2	非形式功能 / 功能块调用
20	1	在 LD 中使用 EN 和 ENO
	2	在 FBD 中不使用 EN 和 ENO
20a	1	In-out 变量声明 (文本)
	2	In-out 变量声明 (图形)
	3	in-out 变量到各种变量 (图形) 的图形连接
21	1	重载功能
	2	典型功能

表格号码	特性号码	特性描述
22	1	*_TO_** (*注释 1， 528 页*)
	2	截断功能 (*注释 2， 528 页*)
	3	*_BCD_TO_** (*注释 3， 529 页*)
	4	**_TO_BCD_* (*注释 3， 529 页*)
23	1	ABS 功能
	2	SQRT 功能
	3	LN 功能
	4	LOG 功能
	5	EXP 功能
	6	SIN 功能
	7	COS 功能
	8	TAN 功能
	9	ASIN 功能
	10	ACOS 功能
	11	ATAN 功能
24	12	ADD 功能
	13	MUL 功能
	14	SUB 功能
	15	DIV 功能
	16	MOD 功能
	17	EXPT 功能
	18	MOVE 功能
25	1	SHL 功能
	2	SHR 功能
	3	ROR 功能
	4	ROL 功能
26	5	AND 功能
	6	OR 功能
	7	XOR 功能
	8	NOT 功能

表格号码	特性号码	特性描述
27	1	SEL 功能
	2a	MAX 功能
	2b	MIN 功能
	3	LIMIT 功能
	4	MUX 功能
28	5	GT 功能
	6	GE 功能
	7	EQ 功能
	8	LE 功能
	9	LT 功能
	10	NE 功能
29	1	LEN 功能 (*注释 4， 529 页*)
	2	LEFT 功能 (*注释 4， 529 页*)
	3	RIGHT 功能 (*注释 4， 529 页*)
	4	MID 功能 (*注释 4， 529 页*)
	6	INSERT 功能 (*注释 4， 529 页*)
	7	DELETE 功能 (*注释 4， 529 页*)
	8	REPLACE 功能 (*注释 4， 529 页*)
	9	FIND 功能 (*注释 4， 529 页*)
30	1a	ADD 功能 (*注释 6， 530 页*)
	1b	ADD_TIME 功能
	2b	ADD_TOD_TIME 功能
	3b	ADD_DT_TIME 功能
	4a	SUB 功能 (*注释 6， 530 页*)
	4b	SUB_TIME 功能
	5b	SUB_DATE_DATE 功能
	6b	SUB_TOD_TIME 功能
	7b	SUB_TOD_TOD 功能
	8b	SUB_DT_TIME 功能
	9b	SUB_DT_DT 功能
	10a	MUL 功能 (*注释 6， 530 页*)
	10b	MULTIME 功能
	11a	DIV 功能 (*注释 6， 530 页*)
	11b	DIVTIME 功能

表格号码	特性号码	特性描述
33	1a	用于内部变量的保留限定词 (*注释 11， 531 页*)
	2a	用于输出变量的保留限定词 (*注释 11， 531 页*)
	2b	用于输入变量的保留限定词 (*注释 11， 531 页*)
	3a	用于内部功能块的保留限定词 (*注释 11， 531 页*)
	4a	VAR_IN_OUT 声明 (文本)
	4b	VAR_IN_OUT 声明和使用 (图形)
	4c	VAR_IN_OUT 声明，带有各种变量的赋值 (图形)
34	1	双稳态功能块 (置位功能)
	2	双稳态功能块 (复位功能)
35	1	上升沿检测器
	2	下降沿检测器
36	1a	CTU(加计数) 功能块
	1b	CTU_DINT 功能块
	1d	CTU_UDINT 功能块
	2a	CTD(减计数) 功能块
	2b	CTD_DINT 功能块
	2d	CTD_UDINT 功能块
	3a	CTUD (加减计数器) 功能块
	3b	CTUD_DINT 功能块
	3d	CTUD_UDINT 功能块
37	1	TP (脉冲) 功能块
	2a	TON (开启延迟) 功能块
	3a	TOF (关闭延迟) 功能块
39	19	使用直接表示的变量
40	1	步和初始步　带有直接链接的图形格式
	3a	步标记 - 一般形式
	4	步实耗时间 - 一般形式

表格号码	特性号码	特性描述
41	7	使用转换名称
	7a	LD 语言中通过转换名称链接的转换条件
	7b	FBD 语言中通过转换名称链接的转换条件
	7c	IL 语言中通过转换名称链接的转换条件
	7d	ST 语言中通过转换名称链接的转换条件
42	1	在一个 VAR 或者 VAR_OUTPUT 功能块中声明的任何一个布尔变量，或者它们的图形化形式，可以是一个动作
	2l	在 LD 语言中动作的图形声明
	2f	在 FBD 语言中动作的图形声明
	3s	在 ST 语言中动作的文本声明
	3i	在 IL 语言中动作的文本声明
43	1	在物理或者逻辑上与步相邻的动作块 (*注释 7， 531 页*)
	2	在物理或者逻辑上与步相邻的毗连动作块 (*注释 7， 531 页*)
44	1	在支持的动作功能块中的动作限定词
	2	在支持的动作功能块中的动作名称
45	1	None - 没有限定词
	2	限定词 N
	3	限定词 R
	4	限定词 S
	5	限定词 L
	6	限定词 D
	7	限定词 P
	9	限定词 DS
	11	限定词 P1
	12	限定词 P0
45a	2	没有 “最终扫描”的动作控制

表格号码	特性号码	特性描述
46	1	单个顺序
	2a	顺序选择的分散：转换运算优先级从左到右
	3	顺序选择的收敛
	4	同步顺序 - 发散和收敛
	5a	顺序跳转：转换运算优先级从左到右
	6a	顺序环路：转换运算优先级从左到右
49	1	CONFIGURATION...END_CONFIGURATION 结构 (*注释 12， 531 页*)
	5a	周期任务结构
	5b	非周期任务结构
	6a	用来关联 PROGRAM 和 TASK 的 WITH 结构 (*注释 9， 531 页*)
	6c	没有 TASK 关联的 PROGRAM 声明 (*注释 10， 531 页*)
50	5a	非抢占调度 (*注释 13， 531 页*)
	5b	抢占调度 (*注释 14， 531 页*)

注释 1

类型变换功能列表：
- BOOL_TO_BYTE, BOOL_TO_DINT, BOOL_TO_INT, BOOL_TO_REAL, BOOL_TO_TIME, BOOL_TO_UDINT, BOOL_TO_UINT, BOOL_TO_WORD, BOOL_TO_DWORD
- BYTE_TO_BOOL, BYTE_TO_DINT, BYTE_TO_INT, BYTE_TO_REAL, BYTE_TO_TIME, BYTE_TO_UDINT, BYTE_TO_UINT, BYTE_TO_WORD, BYTE_TO_DWORD, BYTE_TO_BIT
- DINT_TO_BOOL, DINT_TO_BYTE, DINT_TO_INT, DINT_TO_REAL, DINT_TO_TIME, DINT_TO_UDINT, DINT_TO_UINT, DINT_TO_WORD, DINT_TO_DWORD, DINT_TO_DBCD, DINT_TO_STRING
- INT_TO_BOOL, INT_TO_BYTE, INT_TO_DINT, INT_TO_REAL, INT_TO_TIME, INT_TO_UDINT, INT_TO_UINT, INT_TO_WORD, INT_TO_BCD, INT_TO_DBCD, INT_TO_DWORD, INT_TO_STRING
- REAL_TO_BOOL, REAL_TO_BYTE, REAL_TO_DINT, REAL_TO_INT, REAL_TO_TIME, REAL_TO_UDINT, REAL_TO_UINT, REAL_TO_WORD, REAL_TO_DWORD, REAL_TO_STRING
- TIME_TO_BOOL, TIME_TO_BYTE, TIME_TO_DINT, TIME_TO_INT, TIME_TO_REAL, TIME_TO_UDINT, TIME_TO_UINT, TIME_TO_WORD, TIME_TO_DWORD, TIME_TO_STRING
- UDINT_TO_BOOL, UDINT_TO_BYTE, UDINT_TO_DINT, UDINT_TO_INT, UDINT_TO_REAL, UDINT_TO_TIME, UDINT_TO_UINT, UDINT_TO_WORD, UDINT_TO_DWORD
- UINT_TO_BOOL, UINT_TO_BYTE, UINT_TO_DINT, UINT_TO_INT, UINT_TO_REAL, UINT_TO_TIME, UINT_TO_UDINT, UINT_TO_WORD, UINT_TO_DWORD,
- WORD_TO_BOOL, WORD_TO_BYTE, WORD_TO_DINT, WORD_TO_INT, WORD_TO_REAL, WORD_TO_TIME, WORD_TO_UDINT, WORD_TO_UINT, WORD_TO_BIT, WORD_TO_DWORD
- DWORD_TO_BOOL, DWORD_TO_BYTE, DWORD_TO_DINT, DWORD_TO_INT, DWORD_TO_REAL, DWORD_TO_TIME, DWORD_TO_UDINT, DWORD_TO_UINT, DWORD_TO_BIT,

每一种变换的效果会在基础数据库的帮助文档中给出。

注释 2

截断功能类型列表：
- REAL_TRUNC_DINT, REAL_TRUNC_INT, REAL_TRUNC_UDINT, REAL_TRUNC_UINT

每一种变换的效果会在基础数据库的帮助文档中给出。

注释 3

BCD 转换功能类型列表：

- BCD_TO_INT, DBCD_TO_INT, DBCD_TO_DINT

List of types for BCD conversion function:

- INT_TO_BCD, INT_TO_DBCD, DINT_TO_DBCD

每一种变换的效果会在基础数据库的帮助文档中给出。

注释 4

字符串功能类型列表：

- LEN_INT, LEFT_INT, RIGHT_INT, MID_INT, INSERT_INT, DELETE_INT, REPLACE_INT, FIND_INT

注释 5

一个变量只能映射到与之类型完全相同的直接表示的变量上去。
也就是说， INT 类型的变量只能映射到直接表示的 INT 类型的变量上去。
但是这个规则有一个例外：对于内部字 (%MW<i>)，输入字 (%IW<i>) 和常数字 (%KW<i>) 内存变量来说，允许使用任何声明的变量类型。
允许的映射：

	语法	数据类型	允许的变量类型
内部位	%M<i> or %MX<i>	EBOOL	EBOOL ARRAY [..] OF EBOOL
内部字	%MW<i>	INT	除了以下几种以外，可以使用其他任何类型： ● EBOOL ● ARRAY [..] OF EBOOL
内部双字	%MD<i>	DINT	没有映射，因为 %MW<i> 和 %MD<i> 以及 %MF<i> 之间会发生重叠。
内部实数	%MF<i>	REAL	没有映射，因为 %MW<i> 和 %MD<i> 以及 %MF<i> 之间会发生重叠。
常数字	%KW<i>	INT	除了以下几种以外，可以使用其他任何类型： ● EBOOL ● ARRAY [..] OF EBOOL

	语法	数据类型	允许的变量类型
常数双字	%KD<i>	DINT	没有映射，因为 %KW<i> 和 %KD<i> 以及 %KF<i> 之间会发生重叠。 只有 Premium PLC 含有此类变量。
常数实数	%KF<i>	REAL	没有映射，因为 %KW<i> 和 %KD<i> 以及 %KF<i> 之间会发生重叠。 只有 Premium PLC 含有此类变量。
系统位	%S<i> or %SX<i>	EBOOL	EBOOL
系统字	%SW<i>	INT	INT
系统双字	%SD<i>	DINT	DINT
输入位	%I<i>	EBOOL	EBOOL ARRAY [..] OF EBOOL 只有 Premium PLC 含有此类变量。
输入字	%IW<i>	INT	除了以下几种以外，可以使用其他任何类型： ● EBOOL ● ARRAY [..] OF EBOOL 只有 Premium PLC 含有此类变量。
公共字	%NWi.j.k	INT	INT
拓扑变量	%I..., %Q..., ...	...	相同的类型 (在一些数字 I/O 模块上可以把 %IX<topo> 和 %QX<topo> 对象映射到 EBOOL 数组上去。)
提取位	%MWi.j, ...	BOOL	BOOL

注释 6

在 ST 语言中只有运算符 “+” (用于加)，“-” (用于减)，“*” (用于乘) 和 “/” (用于除)。

注释 7	这个特性只出现在图表的 “扩展视图” 中。
注释 8	这个特性出现在图表的 “扩展视图” 中，但是不是作为毗连的功能块，而是作为一个可滚动的动作名称列表而出现，这些名称在单个功能块符号中带有相关的限定词。
注释 9	只有一个程序实例到任务的一对一映射。文本格式会被一个属性对话框所代替。
注释 10	文本格式被一个属性对话框所代替。
注释 11	所有变量都是保持性的 (在变量声明中隐式地假定了 `RETAIN` 限定词)。
注释 12	文本格式被项目浏览器演示所代替。
注释 13	通过 Mask 指令，用户可以获得一个非抢占的行为。您可以在系统功能库中找到 `MASKEVT` (全局 EVT 掩码) 和 `UNMASKEVT` (全局 EVT 消除掩码)。
注释 14	**在缺省情况下，多重任务系统是抢占性的。**

IL 语言的元素

IL 语言的元素

IL 语言元素的 IEC 遵循性表格：

表格号码	特性号码	特性描述
51b	1	以显式运算符开始的带括号的表达式
51b	2	带括号的表达式 (简易格式)
52	1	LD 运算符 (带有限定词 “N”)
	2	ST 运算符 (带有限定词 “N”)
	3	S， R 运算符
	4	AND 运算符 (带有限定词 “(”， “N”)
	6	OR 运算符 (带有限定词 “(”， “N”)
	7	XOR 运算符 (带有限定词 “(”， “N”)
	7a	NOT 运算符
	8	ADD 运算符 (带有限定词 “(”)
	9	SUB 运算符 (带有限定词 “(”)
	10	MUL 运算符 (带有限定词 “(”)
	11	DIV 运算符 (带有限定词 “(”)
	11a	MOD 运算符 (带有限定词 “(”)
	12	GT 运算符 (带有限定词 “(”)
	13	GE 运算符 (带有限定词 “(”)
	14	EQ 运算符 (带有限定词 “(”)
	15	NE 运算符 (带有限定词 “(”)
	16	LE 运算符 (带有限定词 “(”)
	17	LT 运算符 (带有限定词 “(”)
	18	JMP 运算符 (带有限定词 “C”， “N”)
	19	CAL 运算符 (带有限定词 “C”， “N”)
	20	RET 运算符 (带有限定词 “C”， “N”) (*注释， 532 页*)
	21	) (估算延迟运算)
53	1a	对带有非形式参数列表的功能块进行 CAL 操作
	1b	对带有形式参数列表的功能块进行 CAL 操作
	2	对带有参数加载 / 存储的功能块进行 CAL 操作
	4	通过形式参数列表调用功能
	5	通过非形式参数列表调用功能

注释

仅限于 DFB。

ST 语言的元素

ST 语言的元素

ST 语言元素的 IEC 遵循性表格：

表格号码	特性号码	特性描述
55	1	加括号 (表达式)
	2	功能运算：功能名称 (参数列表)
	3	求幂：**
	4	否定：-
	5	求反：NOT
	6	乘：*
	7	除：/
	8	求模：MOD
	9	加：+
	10	减：-
	11	比较：<, >, <=, >=
	12	等于：=
	13	不等于：<>
	14	布尔与：&
	15	布尔与：AND
	16	布尔异或：XOR
	17	布尔或：OR
56	1	赋值
	2	功能块调用和功能块输出使用
	3	RETURN 指令 (*注释， 533 页*)
	4	IF 指令
	5	CASE 指令
	6	FOR 指令
	7	WHILE 指令
	8	REPEAT 指令
	9	EXIT 指令
	10	Empty 指令

注释

仅限于 DFB。

公共图形元素

公共图形元素

公共图形元素的 IEC 遵循性表格：

表格号码	特性号码	特性描述
57	2	水平线：图形或者半图形
	4	垂直线：图形或者半图形
	6	水平 / 垂直连接：图形或者半图形
	8	线交叉而不连接：图形或者半图形
	10	连接的和非连接的拐角：图形或者半图形
	12	带有连线的功能块：图形或者半图形
58	1	无条件跳转：FBD 语言
	2	无条件跳转：LD 语言
	3	条件跳转：FBD 语言
	4	条件跳转：LD 语言
	5	条件返回：LD 语言 (*注释， 534 页*)
	6	条件返回：FBD 语言 (*注释， 534 页*)
	7	从功能或者功能块的无条件返回 (*注释， 534 页*)
	8	无条件返回：LD 语言 (*注释， 534 页*)

注释

仅限于 DFB。

LD 语言的元素

LD 语言的元素

LD 语言元素的 IEC 遵循性表格：

表格号码	特性号码	特性描述
59	1	左电源母线
	2	右电源母线
60	1	水平链接
	2	垂直链接
61	1	正常打开的触点 (竖线) *(注释， 535 页)*
	3	正常闭合的触点 (竖线) *(注释， 535 页)*
	5	上升沿触点 (竖线) *(注释， 535 页)*
	7	下降沿触点 (竖线) *(注释， 535 页)*
62	1	线圈
	2	否定线圈
	3	SET (锁定) 线圈
	4	RESET (开锁) 线圈
	8	上升沿线圈
	9	下降沿线圈

注释

只有图形表示方法。

与实现过程相关的参数

与实现过程相关的参数

与实现过程相关的参数的 IEC 遵循性表格：

参数	限制 / 行为
标识符的最大长度	32 个字符
注释的最大长度	在 Unity Pro 内：每个编辑器对象为 1024 个字符。 注意：在持久层受到 XML 的限制或者 UDB 串应用的限制。
语言的语法和语义	Unity 只安装了一个语言检查，用于老程序的转换器： `{ ConvError (' error text'); }` 任何其他语言构造的错误都会忽略 (会给出一个警告信息)
当某一个特殊的应用支持表格 5 中的特性 #4，而不支持特性 #2 时，用于双引号字符应用的语法和语义。	(支持表格 5 中的特性 #2)
TIME, DATE, TIME_OF_DAY 和 DATE_AND_TIME 类型变量取值范围和表示的精确度	对 TIME 来说：t#0ms .. t#4294967295ms (=t#49D_17H_2M_47S_295MS) 对 DATE 来说：D#1990-01-01 .. D#2099-12-31 对 TOD 来说：TOD#00:00:00 .. TOD#23:59:59
TIME, TIME_OF_DAY 和 DATE_AND_TIME 类型用秒来表示的精确度	TIME：精确度 1 毫秒 TIME_OF_DAY：精确度 1 秒
枚举值的最大数值	不可用
数组下标的最大数值	6
数组的最大尺寸	64k 字节
结构元素的最大数量	无限制
最大结构尺寸	64k 字节
下标值的最大范围	DINT 的范围
嵌套结构级别的最大数量	10
STRING 和 WSTRING 变量的缺省最大长度	16 个字符
STRING 和 WSTRING 变量的最大允许长度	64k 字节

参数	限制 / 行为
逻辑或者物理映射层次结构的最大数量	Premium：物理映射 (5 级) Quantum：逻辑映射 (1 级)
可扩展功能的最大输入数量	所有输入参数 (包括 in-out 参数) 的数量被限定为 32 个。所有输出参数 (包括 in- out 参数) 的数量也被限定为 32 个。 因而可扩展输入参数的限制为 (32 个 - 输入参数的数量＋ in-out 参数的数量)。 因而可扩展输出参数的限制为 (32 个 - 输出参数的数量 ＋ in-out 参数的数量)。
类型转换对精确度的影响	参见联机帮助。
在类型转换中的出错条件	在联机帮助中有针对出错条件的描述。在全局中，%S18 针对溢出错误而置位。 ENO 也会置位。结果取决于具体的功能。
数字功能的精确度	INTEL 浮点处理或者仿真。
在时间数据类型和其他在表格 30 中定义的数据类型之间的类型转换对精确度的影响	参见联机帮助。
功能块规范和实例化的最大数量	只受代码段最大尺寸的限制。
当 EN 为 FALSE 时的功能块输入变量赋值	没有赋值
计数器的 Pvmin， Pvmax	INT 基数计数器： ● Pvmin=-32768 (0x8000) ● Pvmax=32767 (0x7FFF) UINT 基数计数器： ● Pvmin=0 (0x0) ● Pvmax=65535 (0xFFFF) DINT 基数计数器： ● Pvmin= -2147483648 (0x80000000) ● Pvmax=2147483647 (0x7FFFFFFF) UDINT 基数计数器： ● Pvmin=0 (0x0) ● Pvmax=4294967295 (0xFFFFFFFF)
在一个计时操作中因输入 PT 数值变化而造成的影响	新的 PT 数值会马上被采用，即使是计时操作正在进行，也会马上采用新的数值

参数	限制 / 行为
程序大小的限制	取决于控制器类型和内存
步实耗时间的精确度	10 毫秒
每个 SFC 的最大步数量	每个 SFC 代码段 1024 个步
转换每个 SFC 和每个步的最大转换数量	受输入步 / 转换的可用区域以及每个 SFC 代码段最大步数量 (1024 个步) 的限制。 每个步 32 个转换。受输入选择 / 并行分支的可用区域的限制，最多 32 行。
每个步动作功能块的最大数量	20
访问 Q 或者 A 输出的功能对等项	不可用
转换清除时间	视目标而定； 总是 < 100 微秒
发散 / 收敛构造的最大宽度	32
RESOURCE 库的内容	不可用
使用 READ_WRITE 访问功能块输出的效果	不可用
任务的最大数量	取决于控制器的类型。 功能最强大的控制器上的任务最多：9 个任务
任务间隔时间分辨率	10 毫秒
表达式的最大长度	实际上没有限制
指令的最大长度	实际上没有限制
CASE 选择的最大数量	实际上没有限制
中止 FOR 循环的控制变量数值	未定义
网络拓扑的限制	没有限制
反馈环的运算顺序	连接到反馈变量的功能块首先执行

出错条件

出错条件

出错条件的 IEC 标准表格：

出错条件	**处理 (参见注释，*540* 页)**
嵌套的注释	2) 在编程的时候会报告错误
某个变量的数值超过了指定的子范围	不可用
某个不完整地址说明的配置丢失 (“*”号)	不可用
试图通过一个程序组织单元修改一个已经被声明为常数的变量	2) 在编程的时候会报告错误
在功能中以不正确的方式使用直接表示的变量或者外部变量	不可用
某个 VAR_IN_OUT 变量没有 “正确映射”	2) 在编程的时候会报告错误
类型转换错误	4) 在执行的时候会报告错误
数字结果超过了数据类型的范围	4) 在执行的时候会报告错误
除零	4) 在执行的时候会报告错误
某一个选择的功能中混合输入数据类型	2) 在编程的时候会报告错误
结果超过了数据类型的范围	4) 在执行的时候会报告错误
某个 in-out 变量没有指定数值	2) 在编程的时候会报告错误
在 SFC 网络中有零个或者多个初始步	
用户程序试图修改步状态或时间	
在计算转换条件时出现副作用	3) 在分析 / 加载 / 链接的时候会报告错误
动作控制竞争错误	不可用
一个选择发散的转换同时为真，没有区分先后顺序	不可用
不安全的或者无法实现的 SFC	3) 在分析 / 加载 / 链接的时候会报告错误
在 VAR_ACCESS 中的数据类型冲突	不可用
某个任务未能按时完成，或者过了执行期限	4) 在执行的时候会报告错误
数字结果超过了数据类型的范围	4) 在执行的时候会报告错误

出错条件	处理 (参见注释， *540* 页)
当前结果和操作数不是同一数据类型	2) 在编程的时候会报告错误
除零	4) 在执行的时候会报告错误
数字结果超过了数据类型的范围	4) 在执行的时候会报告错误
运算中出现了无效的数据类型	4) 在执行的时候会报告错误
从功能返回时没有赋值	不可用
循环未能中止	4) 在执行的时候会报告错误
把同一个标识符用作连接器标记和元素名称	不可用
未初始化的反馈变量	1) 不报告错误

注释

根据 IEC 61131-3 子条款 1.5.1， d)，针对各种出错条件的识别处理：

- 不报告错误
- 在编程的时候报告错误
- 在分析 / 加载 / 链接的时候报告错误
- 在执行的时候报告错误

A.3 IEC 61131-3 的扩展内容

IEC 61131-3 第 2 版的扩展内容

内容预览

除了前面所列出的标准 IEC 特性 (参见 *IEC 遵循性表格， 519 页*) 以外， Unity Pro 编程环境还从 PL7 编程环境集成了众多的特性。用户可以通过相应的选项对话对这些扩展内容进行选择。关于对话和特性的详细内容，会在联机帮助中的*数据和语言*一章给出。

还有另外一个扩展内容不在选项对话中，该内容是从 PL7 和 Concept 编程环境继承过来的：Unity Pro 在所有编程语言中都提供了称为代码段的构造，借助代码段，用户可以对程序组织单元 (POU) 进行进一步划分。通过这种构造，用户可以在一个 POU 体中溶合若干种语言 (比如 FBD 代码段， SFC 代码段)，在这种情况下，该特性扩展了 IEC 的语法。一个遵循 IEC 标准的 POU 体应该只包含一个单一的代码段。代码段并没有创建一个独立的命名适用范围；所有语言元素的命名适用范围都是 POU。 .

代码段的用途

代码段有多种用途：

- 通过代码段，用户可以从功能的角度把较大的 POU 体进行进一步划分：用户可以把 POU 体划分为若干有意义的功能块。代码段的列表代表了一个较大的，或者非结构化的 POU 体内容的一类功能表格。
- 通过代码段，用户可以从图形的角度把较大的 POU 体进行进一步划分：用户可以根据图形表示的需要来设计 POU 体的子级结构。用户可以创建自己所喜欢的小型或大型图形代码段。
- 通过把较大的 POU 体进一步细分，用户可以进行快速的联机修改操作：用户可以在 Unity Pro 中以代码段为单位来进行联机更改。在运行时中，如果在不同的位置对 POU 体进行了修改，那么只要给出明确的请求，受到改动影响的所有代码段都会自动被下载。
- 通过代码段，用户可以重新安排 POU 体中那些带有标记的特定部分的执行顺序：位于代码段中的代码段名称会充当该部分在 POU 体中的标记。用户只要对这些标记做出相应处理，就可以安排相应部分的执行顺序。
- 通过代码段，用户可以在同一个 POU 中同时使用多种语言：该特性是 IEC 语法扩展的主要内容，而 IEC 只允许在一个 POU 体中使用一种 IEC 语言。在一个遵循 IEC 标准的程序体中，必须使用 SFC 来管理程序体中的各种语言 (每一种转换和动作都可以用自己的语言来表达)。

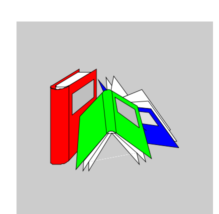
A.4 文本语言的语法

文本语言的语法

描述

Unity Pro 编程环境尚不支持对遵循 IEC 61131-3 第 2 版附录 B 全文本语言语法的文本文件进行导入或者导出操作。
不过，文本语言代码段支持在 IEC 61131-3 第 2 版附录 B.2 和 B.3 中所说明的 IL 和 ST 语言文本语法，其中包括所有直接和间接引用的附录 B.1 的内容。
根据遵循性表格 (参见 *IEC 遵循性表格，519 页*) 给出的特性，IEC 61131-3 第 2 版附录 B 中不被 Unity Pro 支持的语法内容没有被应用。

术语表



!

%I 根据 IEC 标准， %I 表示一个离散量输入类型的语言对象。

%IW 根据 IEC 标准， %IW 表示一个模拟量输入类型的语言对象。

%KW 根据 IEC 标准，%KW 表示一个常数字类型的语言对象。

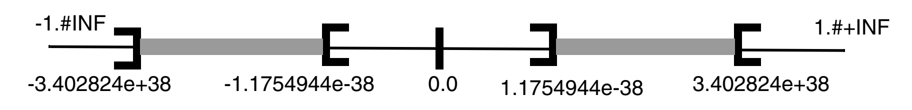
%M 根据 IEC 标准，%M 表示一个内存位类型的语言对象。

%MW 根据 IEC 标准，%MW 表示一个内存字类型的语言对象。

%Q 根据 IEC 标准，%Q 表示一个离散量输出类型的语言对象。

%QW 根据 IEC 标准， %QW 表示一个模拟量输出类型的语言对象。

1.#INF 用来表示一个数值超过了允许的范围。
对于一个整数来说，数值范围 (灰色) 如下所示：



当一个计算结果：
- 小于 -3.402824e+38，就会显示 -1.#INF 符号 (负无穷)，
- 大于 +3.402824e+38，就会显示 1.#INF 符号 (正无穷)。

1.#NAN 用来表示运算的结果不是一个数值 (NAN = 不是一个数值)。
例子：计算负数的平方根。



Animating the links 对链接进行动态化显示 也称为**电源流**，表示在梯形图语言和功能块中所使用的一类动态操作。根据所连接的变量的类型，链接会以红色，绿色或者黑色显示出来。

ANY 各种数据类型之间有一个层次结构。在 DFB 中，有的时候可以声明哪些变量能够包含若干类数值。在这种情况下，我们使用 ANY_xxx 类型。

下面的图表显示了层次结构：

```
ANY
  ANY_ELEMENTARY
    ANY_MAGNITUDE_OR_BIT
      ANY_MAGNITUDE
        ANY_NUM
          ANY_REAL
            REAL
          ANY_INT
            DINT, INT, UDINT, UINT
        TIME
      ANY_BIT
        DWORD, WORD, BYTE, BOOL
    ANY_STRING
      STRING
    ANY_DATE
      DATE_AND_TIME, DATE, TIME_OF_DAY
    EBOOL
  ANY_DERIVED
    ANY_ARRAY
      ANY_ARRAY_ANY_EDT
        ANY_ARRAY_ANY_MAGNITUDE
          ANY_ARRAY_ANY_NUM
            ANY_ARRAY_ANY_REAL
              ANY_ARRAY_REAL
            ANY_ARRAY_ANY_INT
              ANY_ARRAY_DINT
              ANY_ARRAY_INT
              ANY_ARRAY_UDINT
              ANNY_ARRAY_UINT
          ANY_ARRAY_TIME
        ANY_ARRAY_ANY_BIT
          ANY_ARRAY_DWORD
          ANY_ARRAY_WORD
          ANY_ARRAY_BYTE
          ANY_ARRAY_BOOL
        ANY_ARRAY_ANY_STRING
          ANY_ARRAY_STRING
        ANY_ARRAY_ANY_DATE
          ANY_ARRAY_DATE_AND_TIME
          ANY_ARRAY_DATE
          ANY_ARRAY_TIME_OF_DAY
        ANY_ARRAY_EBOOL
      ANY_ARRAY_ANY_DDT
    ANY_STRUCTURE
      ANY_DDT
      ANY_IODDT
      ANY_FFB
        ANY_EFB
        ANY_DFB
```

ARRAY 数组

数组是一个同类元素的数据表。
它的语法如下：数组 [< 范围 >] OF < 类型 >
例子：
`ARRAY [1..2] OF BOOL` 是由两个 `BOOL` 类型元素组成的一维数据表。
`ARRAY [1..10, 1..20] OF INT` 是由 10x20 个 `INT` 类型元素组成的二维数据表。

ASCII

ASCII 是美国信息交换标准码的简称。该美国编码 (业已成为国际标准) 用 7 个位来定义在英语，标点符号，特定图形字符以及其他混合命令中所使用的所有字母数字字符。

Auxiliary tasks 辅助任务

可选的周期任务，用来处理不需要进行快速操作的功能程序：测量，调整，诊断帮助，等等。

B

Base 10 literals 基数为 10 的字面值

基数为 10 的字面值用来表示一个十进制整数。这个数值前面可以带有 “+” 号和 “-” 号。如果在这个字面值中使用了 “_” 字符，该字符不会对该字面值造成影响。
例子：
`-12, 0, 123_456, +986`

Base 16 literals 基数为 16 的字面值

基数为 16 的字面值用来表示一个十六进制整数。基数是由数值 “16” 和 “#” 号决定的。不允许使用 “+” 号和 “-” 号。为了阅读方便，您可以在各个位之间使用 “_” 字符。
例子：
`16#F_F` 或 `16#FF` (对应十进制的 255)
`16#F_F` 或 `16#FF` (对应十进制的 224)

Base 2 literals 基数为 2 的字面值

基数为 2 的字面值用来表示一个二进制整数。基数是由数值 “2” 和 “#” 号决定的。不允许使用 “+” 号和 “-” 号。为了阅读方便，您可以在各个位之间使用 “_” 字符。
例子：
`2#1111_1111` 或 `2#11111111` (对应十进制的 255)
`2#1110_0000` 或 `2#11100000` (对应十进制的 224)

Base 8 literals 基数为 8 的字面值

基数为 8 的字面值用来表示一个八进制整数。基数是由数值 “8”和 “#”号决定的。不允许使用 “+”号和 “-”号。为了阅读方便，您可以在各个位之间使用 “_”字符。
例子：
8#3_77 或 *8#377* (对应十进制的 255)
8#34_0 或 8#340 (对应十进制的 224)

BCD

二进制编码的十进制 (BCD) 格式用四位 (半字节) 来表示 0 到 9 之间的十进制数。在这个格式中，用来对十进制数进行编码的四个位的组合会出现闲置的情况。
BCD 编码的例子：
- 数值 2450
- 被编码为：0010 0100 0101 0000

BIT 位

用来表示信息数量的二进制单元，可以表示两个数值 (或者状态)：0 或 1。

BOOL 布尔

BOOL 是布尔类型的简写。它是运算中用到的基本数据项。BOOL 类型变量的值为：0 (FALSE) 或 1 (TRUE)。比如一个 BOOL 类型的字提取位：%MW10.4。

Break point 断点

用在应用程序的 “调试”模式中。
它是唯一的 (在某一时间)，在达到断点以后，系统会向处理器发送信号，停止程序的运行。
它可用于在线模式，可以在如下程序元素中进行定位：
- LD 网络，
- 结构化文本序列或者指令表，
- 结构化文本的行 (行模式)。

BYTE 字节

8 位在一起构成一个字节。字节以二进制或者八进制的格式输入。
字节类型以 8 位格式编码，它对应十六进制从 16#00 到 16#FF 的范围。

C

Constants 常数

位于常数区 (%K) 的 INT， DINT 或 REAL 类型变量，或者在直接寻址 (%KW， %KD 或 %KF) 中使用的变量。在执行程序的过程中不能修改这些常数。

CPU

它是中央处理器的简称。
它是一个微处理器，由控制单元和算术单元组成。控制单元的作用是从中央内存中提取要执行的指令，以及执行该指令所需的数据，在算术和逻辑单元之间建立电连接，并在该单元中对数据进行处理。某些情况下，在同一个芯片上还会包括 ROM 或者 RAM 内存，甚至 I/O 接口或者缓冲器。

Cyclic execution 循环执行

主任务以循环或者周期的方式执行。在循环执行中，周期是连续的，在相邻的周期之间没有等待时间。



DATE

`DATE` 类型以 32 的 BCD 格式编码，它包含如下信息：

- 在一个 16 位区中编码的年，
- 在一个 8 位区中编码的月，
- 在一个 8 位区中编码的日。

`DATE` 类型按照如下格式输入：**D#**< 年 >-< 月 >-< 日 >

下面的表格给出了每一区的下限 / 上限：

区	限定范围	注释
年	[1990,2099]	年
月	[01,12]	左面的 0 会一直显示出来，但是在输入时间的时候可以忽略
Day	[01,31]	对 01\03\05\07\08\10\12 月
	[01,30]	对 04\06\09\11 月
	[01,29]	对 02 月 (闰年)
	[01,28]	对 02 月 (非闰年)

DATE_AND_TIME

参见 `DT`

DBCD

表示一个 BCD 格式的双整数。

二进制编码的十进制 (BCD) 格式用四位 (半字节) 来表示 0 到 9 之间的十进制数。在这个格式中，用来对十进制数进行编码的四个位的组合会出现闲置的情况。

DBCD 编码的例子：

- 数值 `78993016`
- 被编码为：`0111 1000 1001 1001 0011 0000 0001 0110`

DDT

DDT 是导出数据类型的缩写。

导出数据类型是一组同一类型的元素 (数组)，或者一组不同类型的元素 (结构)

DFB

DFB 是导出功能块的缩写。

DFB 类型是可以用 ST， IL， LD 或者 FBD 编写的功能块。

通过在应用程序中使用 DFB 类型，用户可以：
- 简化程序的设计和输入，
- 增加程序的可读性，
- 方便程序调试，
- 减小生成代码的大小。

DFB 实例

在从一个语言编辑器内调用一个实例时，会生成 DFB 类型实例。实例拥有一个名称，输入 / 输出接口，公共和私有变量会被复制 (每个实例有一个副本，代码不被复制)。DFB 类型可以有多个实例。

DINT

DINT 是双整数格式的缩写 (以 32 位编码)。
它的下限和上限为：-(2 的 31 次方) 到 (2 的 31 次方) -1。
例子：
-2147483648, 2147483647, 16#FFFFFFFF。

Documentation (文档)

包含所有项目信息。在经过编辑以后，文档可以打印出来，用于维护。
文档中的信息包括：
- 硬件和软件配置，
- 程序，
- DFB 类型，
- 变量和动态数据表，
- 交叉引用。
- ...

在创建文档文件的时候，您可以选择以上内容的全部或者一部分。

Driver (驱动程序)

用来将外围设备的存在和特性告知用户计算机操作系统的程序。我们也使用 “外围设备驱动程序”这一术语。为了在 PLC 和个人电脑之间进行通信，用户需要安装驱动程序 (Uni-Telway， XIP， Fipway，等等)。

DT

DT 是日期和时间的缩写。
DT 类型用 64 位的 BCD 格式编码，它包含以下信息：
- 在一个 16 位区中编码的年，
- 在一个 8 位区中编码的月，
- 在一个 8 位区中编码的日，
- 在一个 8 位区中编码的点钟，
- 在一个 8 位区中编码的分，
- 在一个 8 位区中编码的秒。

注意：8 个最低位没有使用。

DT 类型按照如下格式输入：

DT#< 年 >-< 月 >-< 日 >-< 点钟 >：< 分 >：< 秒 >

下面的表格给出了每一栏的下限 / 上限：

栏	限定范围	注释
年	[1990,2099]	年
月	[01,12]	左面的 0 会一直显示出来，但是在输入时间的时候可以忽略
日	[01,31]	对 01\03\05\07\08\10\12 月
	[01,30]	对 04\06\09\11 月
	[01,29]	对 02 (闰年) 月
	[01,28]	对 02 (非闰年) 月
点钟	[00,23]	左面的 0 会一直显示出来，但是在输入时间的时候可以忽略
分	[00,59]	左面的 0 会一直显示出来，但是在输入时间的时候可以忽略
秒	[00,59]	左面的 0 会一直显示出来，但是在输入时间的时候可以忽略

DWORD

DWORD 是双字的缩写。

DWORD 类型以 32 位格式编码。

下面的表格给出了可以使用的基数对应的上限 / 上限：

基数	下限	上限
十六进制	16#0	16#FFFFFFFF
八进制	8#0	8#37777777777
二进制	2#0	2#11111111111111111111111111111111

演示的例子：

数据内容	各个基数的表示形式
00000000000010101101110011011110	16#ADCDE
00000000000000010000000000000000	8#200000
00000000000010101011110011011110	2#10101011110011011110

E

EBOOL

EBOOL 是扩展布尔类型的缩写。它可以用来处理上升沿或下降沿，以及进行强制操作。

一个 EBOOL 类型的变量在内存中占据一个字节。

EDT

EDT 基本数据类型的缩写。

这些类型包括：
- BOOL,
- EBOOL,
- WORD,
- DWORD,
- INT,
- DINT,
- UINT,
- UDINT,
- REAL,
- DATE,
- TOD,
- DT.

EF

是基本功能的缩写。
它是在程序中所使用的模块，执行一个预先定义的软件功能。
功能没有内部状态信息。使用同样的输入参数多次调用同一功能，会给出相同的输出值。关于功能调用的详细图形信息可以在 “[功能块 (实例)] ”中找到。与功能块调用相比，功能调用只有一个未命名的输出，该输出采取与功能相同的名称。在 FBD 中，每一次调用都会通过图形功能块以一个唯一的号码来表示。这个号码是自动生成的，不能改变。
您可以在自己的应用程序中根据需要来放置和设定这些功能。
您也可以使用 SDKC 开发包来开发其他功能。

EFB

是基本功能块的缩写。
它是在程序中所使用的功能块，执行一个预先定义的软件功能。
EFB 具有内部状态和参数。即使输入相同，输出值也可能有所不同。比如，计数器有一个表示已经达到预设值的输出。当当前值等于预设值时，输出就置为 1。

基本功能

参见 EF

EN / ENO
使能 / 出错通知

EN 表示**使能**，它是一个可选的功能块输入。
如果 EN = 0，功能块没有激活，它的内部程序没有执行， ENO 置为 0。
如果 EN = 1，功能块的内部程序执行， ENO 被系统置为 1。如果发生了错误， ENO 会置为 0。
ENO 表示**出错通知**，它是与可选的 EN 输入相关的输出。如果 ENO 置为 0 (因为 EN=0 或者执行的过程中发生错误)，
- 功能块的输出保持上个扫描周期正确执行时的状态
- 功能和功能程序的输出置为 “0”。

Event processing
事件处理

事件处理是由一个事件启动的程序代码段。在处理器接收到了一个软件应用程序事件 (定时器) 或者一个硬件事件 (应用程序专用模块) 以后，在该代码段中编写的指令就会被执行。
事件处理比其他任务的优先级都高，系统一旦检测到了事件，就会马上处理。
事件处理 EVT0 具有最高优先级。所有其他事件处理的优先级相同。

F

Fast task
快速任务

以周期方式执行的任务 (在配置中进行周期设置)，用来执行优先级比主 (master) 任务高的一部分应用程序。

FBD

FBD 是功能块的缩写。
FBD 是一种图形编程语言，以逻辑图表的形式进行操作。除了简单的逻辑功能块 (AND， OR， 等等) 以外，程序的每个功能或功能块也都用这种图形格式来表示。每一个功能块的输入都位于左侧，输出都位于右侧。功能块的输出可以链接到其他功能块的输入，以形成复杂表达式。

FFB

EF (基本功能)， EFB (基本功能块) 和 DFB (导出功能块) 的统称。

Flash Eprom

PCMCIA 存储卡含有程序以及应用程序的常数。

FNES

FNES 是费希尔中性输入输出的缩写。
FNES 格式描述了使用机架，模块和通道的 PLC 的树性结构。
它基于 CNOMO 标准 (机床工具设备标准化委员会)。

Function 功能 参见 EF

Function block 功能块 参见 EFB

Function view 功能视图 通过功能视图，用户可以通过自己创建的功能模块来浏览应用程序中的程序部分 (参见功能组件定义)。

Functional Module 功能组件 功能组件是一组程序元素 (代码段，子程序，宏步，动态数据表，操作员界面，等等)，它的作用是执行一个自动化设备功能。
功能组件可以进一步划分为低级的功能模块，它们用来执行自动化设备主功能的一个或多个子功能。

G

GRAY 格雷码 格雷码或者 “反射二进制”码用来把一个数值编码发展成一个二进制链，这种配置使得以一为变化的值，仅通过一位的变化来表示。
这种代码可以用来避免以下突发事件：在纯粹的二进制中，如果数值 0111 变为 1000，可能会产生 0 到 1000 之间的随机数，因为各个位并不是同时发生变化的。
二进制， BCD 和格雷码之间的比较：

十进制	0	1	2	3	4	5	6	7	8	9
二进制	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
格雷码	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101

H

Hyperlink (超链接)

通过超连接功能，用户可以在项目和外部文档之间建立链接。用户可以在项目目录，变量，处理页面对象等内容的所有元素中创建超链接。外部文档可以是网页，文件 (xls， pdf， wav， mp3， jpg， gif，等等)。

I

IEC 61131-3

国际标准：可编程逻辑控制第 3 部分：编程语言。

IL

IL 是指令表的缩写。

该语言包含一系列基本指令。

该语言与用来对处理器进行编程的汇编语言非常接近。每一个指令都包含一个指令代码和一个操作数。

Instanciate 实例化

对一个对象进行实例化，就是要为它分配一个内存空间，这个内存空间的大小取决于要实例化的对象的类型。当一个对象被实例化以后，它就存在在了，可以被程序进行操作。

INT

`INT` 是单整数格式的缩写 (用 16 位编码)。

它的下限和上限为：-(2 的 31 次方) 到 (2 的 31 次方) - 1.

例子：

`-32768, 32767, 2#1111110001001001, 16#9FA4.`

Integer literals 整数的字面值

整数的字面值用来在十进制系统中输入整数值。这些值的前面可以带有一个符号 (+/-)。在各个数值之间如果有一个下化线 (_)，不会对数值造成影响。

例子：

`-12, 0, 123_456, +986`

IODDT

IODDT 是输入 / 输出导出数据类型的缩写。

术语 IODDT 表示一种结构化数据类型，它代表 PLC 模块的一个模块或通道。每一个应用专家模块都拥有自己的 IODDT。

K

Keyword 关键字 关键字是一个唯一的字符组合，它用作符合语法的编程语言元素 (参见 附录 B IEC 标准 61131-3 定义)。在 Unity Pro 和这个标准中所使用的所有关键字都在 IEC 标准 61131-3 的附录 C 中列出。这些关键字在您的程序中不能用作标识符 (变量名称，代码段， DFB 类型，等等)。

L

LD LD 是梯形图的缩写。
LD 是一种编程语言，它的指令以图形图表的形式执行，这些图表与机床电路图非常类似 (接触，线圈，等等)。

Located variable 定位型变量 定位变量使用户能够知道它在 PLC 内存中的位置。比如，变量 `Water_pressure` 与 `%MW102` 相关联。 `Water_pressure` 称为本地化变量。

M

Macro step 宏步 宏步是一组步和转换的符号表示，它以一个输入步开头，以一个输出步结尾。
宏步之间可以彼此调用。

Master task 主任务 主程序任务。
它是强制性的，用来执行 PLC 的顺序处理。

Mono Task 单任务 由一个单一的任务所组成的应用程序，该任务必须是主任务。

Multi task 多任务 由若干个任务组成的应用程序 (主任务，快速任务，辅助任务，事件处理)。执行这些任务的优先级由 PLC 操作系统来定义。

Multiple token 多令牌 SFC 的操作模式。在多令牌模式下， SFC 可以在同一时间处理多个有效步。

N

Naming convention (identifier) 命名惯例 (标识符)

标识符是一个字符序列，由字母，数字和下化线组成，以字母或者下划线开头 (比如一个功能块类型的名称，一个实例，一个变量或者一个代码段)。除了在项目名和 DFB 名称中以外，都可以使用国家字符集中的字符 (比如：ö,ü, é, õ) 在标识符中，下划线是有效的，比如 `A_BCD` 和 `AB_CD` 会被当作不同的标识符。多个开始下划线和连续下划线是无效的。

标识符不能包含空格，它不区分大小写，比如 `ABCD` 和 `abcd` 会被当作同一个标识符。根据 IEC 61131-3，不能用数字作标识符的起始位。不过，如果您把**语言扩展内容**选项卡内工具 → **项目设置**对话的**起始数字位**激活，那么就可以使用这个功能。

标识符不能是关键字。

Network 网络

网络主要用于通信，它是一组互相通信的工作站。网络这个术语也用来定义一组互连的图形元素。这组元素构成一个程序的一部分，该程序可能由一组网络所构成。

O

Operator screen 操作屏

它是一个集成在 Unity Pro 中的编辑器，用来为自动化进程操作提供方便。用户调试和监视工艺操作，如果出现任何问题，能够快速方便地进行处理。



P

Periodic execution 周期执行

主任务可以以循环或者周期方式来执行。在周期模式下，您要确定一个具体的时间 (周期)，在该时间内，主任务必须被执行。如果在该时间内能够执行主任务，在下一个周期之前就会生成一个等待时间。如果执行超时，那么控制系统就会提示溢出。如果溢出程度过高，PLC 就会停止。

Procedure 功能程序

功能程序是着眼于技术角度的功能视图。它与基本功能的唯一区别就是功能程序可以有多个输出，并且支持 `VAR_IN_OUT` 数据类型。除此之外，功能程序与基本功能没有区别。

功能程序是 IEC 61131-3 的补充内容。

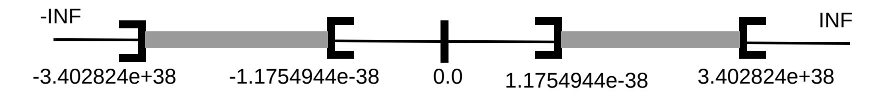
Protection 保护

保护程序元素不能读出 (读保护)，写入或者修改 (读 / 写保护) 的选项。
保护功能通过一个口令进行确认。

R

REAL 实数

实数类型以 32 位编码。
数值的范围在下面用灰色表示出来：



当一个计算结果：
- 在 -1.175494e-38 和 1.175494e-38 之间，它近似为 0，
- 小于 -3.402824e+38，会显示 `-INF` 符号 (负无穷)，
- 大于 +3.402824e+38，会显示 `INF` 符号 (正无穷)，
- 未定义 (负数的平方根)，会显示 `-NAN` 或 `NAN`。

Real literals 实数字面值

实数字面值是用一个或多个十进制数来表示的数值。
例子：
`-12.0`, `0.0`, `+0.456`, `3.14159_26`

Real literals with exponent 带有指数的实数字面值

十进制的字面值可以用下面所示的标准科学计数法表示出来：尾数 + 指数。
例子：
`-1.34E-12` 或 `-1.34e-12`
`1.0E+6` 或 `1.0e+6`
`1.234E6` 或 `1.234e6`

RS 232C

串行通信标准，定义了以下服务的电压：
- 一个 +12 V 的信号表示一个逻辑 0，
- 一个 -12 V 的信号表示一个逻辑 1。

在信号变弱的情况下，检测范围最多可以到 -3 V 和 +3 V。
在这两个限定范围之间，信号被认为是无效的。
RS 232 连接对干扰非常敏感。标准规定不能超过 15 m 的距离，或者 9600 波特 (位 / 秒)。

RS 485	在 10 V/+5 V 差分操作的串行连接标准。它使用两条线来发送 / 接收。它们可以通过 “3 态” 在中止发送以后输出切换到倾听模式。
RUN 运行	激活 PLC 应用程序启动操作的功能。
RUN Auto 自动运行	在冷启动时，激活 PLC 应用程序自动启动功能的执行。
Rung 梯级	梯级相当于 LD 中的序列，另外两个相关的术语是 “梯形网络” 或者更为常规的叫法 “网络”。梯级由 LD 编辑器在两条母线之间形成，它包含一组通过水平或垂直连线互连的图形元素。梯级的大小为 17 到 256 行， 11 到 64 列。

S

Section 代码段	隶属于一个任务的程序模块，编程者可以自己选择编写代码段的语言 (FBD， LD， ST， IL，或者 SFC)。 一个任务可以含有多个代码段，代码段的执行顺序与它们创建和修改的顺序相当。
SFC	SFC 是顺序功能图的简称。 通过 SFC，用户可以用图形化和结构化的方式来表示顺序的自动化设备。这种用图形方式表示自动化设备顺序行为的操作，以及由它所产生的各种结果，都通过简单图形符号来实现。
SFC objects SFC 对象	SFC 对象是一个代表顺序图表动作或转换的状态属性的数据结构。
Single token 单令牌	一个 SFC 图表的操作模式，在该模式下，在同一时间只有一个有效步。
ST	ST 是结构化文本语言的缩写。 结构化文本语言是一种与计算机编程语言类似的复杂语言。通过它，用户可以构建指令序列。
STRING 字符串	字符串类型的变量是一个标准的 ASCII 字符串。一个字符串的长度最多为 65534 个字符。

Structure 结构 在项目浏览器中的视图表示项目结构。

Subroutine 子程序 隶属于一个任务 (主任务，快速任务) 的程序模块，编程者可以选择编写子程序的语言 (FBD，LD， ST 或者 IL)。
一个子程序只能被一个代码段或者隶属于声明它的任务的另外一个子程序所调用。

T

Task 任务 一组代码段和子程序，对 MAST 任务来说以周期或循环的方式执行，对 FAST 任务以周期方式执行。
任务具有一个优先级级别，并且被链接到 PLC 的输入和输出。这些 I/O 马上就会更新。

TIME 时间 TIME 类型以毫秒的形式来表示一个持续时间。它用 32 位来编码，这种数据类型得用户能够获得 0 到 (2 的 32 次方)-1 毫秒的扫描时间。

Time literals 时间字面值 TIME 类型有以下单位：日 (d)，时 (h)，分 (m)，秒 (s) 和毫秒 (ms)。TIME 类型的字面值由 T# t#， TIME# 或者 time# 这些前置字符组合来表示。
例子：T#25h15m, t#14.7S, TIME#5d10h23m45s3ms

Time Out 超时时间 在通信项目中，超时时间是指当超过这个延迟时间，如果目标设备没有响应，通信就会停止。

TIME_OF_DAY 日时间 参见 TOD

TOD

TOD 是日时间的缩写。
TOD 类型以 32 位格式的 BCD 编码，它包含如下信息：
- 在一个 8 位区编码的时，
- 在一个 8 位区编码的分，
- 在一个 8 位区编码的秒。

> 注意：最低的 8 位没有使用。

日时间类型按照如下格式输入：**TOD#**< 时 >：< 分 >：< 秒 >
下面的表格给出了每栏的下限 / 上限：

栏	限定范围	注释
时	[00,23]	左面的 0 会一直显示出来，但是在输入时间的时候可以忽略
分	[00,59]	左面的 0 会一直显示出来，但是在输入时间的时候可以忽略
秒	[00,59]	左面的 0 会一直显示出来，但是在输入时间的时候可以忽略

例子：`TOD#23:59:45`.

Token 令牌 SFC 的一个有效步称为一个令牌。

U

UDINT `UDINT` 是无符号双整数格式的缩写 (以 32 位编码)。它的下限和上限为：0 到 (2 的 32 次方) -1。
例子：
`0,4294967295,2#11111111111111111111111111111111,8#37777777777,16#FFFFFFFF`。

UINT `UINT` 是无符号整数格式的缩写 (以 16 位编码)。它的下限和上限为：0 到 (2 的 16 次方) -1。
例子：
`0,65535,2#1111111111111111,8#177777,16#FFFF`。

Unlocated variable 非定位型变量 非定位型变量是用户无法在 PLC 内存确认其位置的变量。没有分配地址的变量被称为非定位型的。



Variable 变量 `BOOL`，`WORD`，`DWORD` 等数据类型的内存实体，它的内容可以在执行程序的时候进行修改。

Visualization window 可视化窗口 该窗口也称为监视窗口，它显示在语言编辑器中无法动态化的变量。只有在某一时间在编辑器内可见的窗口会被显示出来。



W

Watch point 观察点

用于应用程序的 “调试”模式。
通过观察点，用户可以在执行程序元素 (包含观察点) 的同时显示动态化的变量，以便于在程序的某一点准确观察它们的数值。

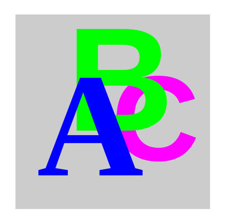
WORD 字

WORD 类型以 16 位格式编码，它用来对位串进行处理。
下面的表格给出了各个基数相应的下限 / 上限：

基数	下限	上限
十六进制	16#0	16#FFFF
八进制	8#0	8#177777
二进制	2#0	2#1111111111111111

表示的例子

数据内容	各个基数的表示
0000000011010011	16#D3
1010101010101010	8#125252
0000000011010011	2#11010011



索引

符号

数字

A

B

C

D

E

F

G

H

I

N

O

P

R

S

T

U

V

W

X

Y

施耐德电气公司
Schneider Electric China
www.schneider-electric.com.cn

北京市朝阳区将台路 2 号
和乔丽晶中心施耐德大厦
邮编: 100016
电话: (010) 8434 6699
传真: (010) 8450 1130

Schneider Building ,Chateau Regency,
No.2 Jiangtai Road, Chaoyang District,
Beijing 100016 China.
Tel: (010) 8434 6699
Fax: (010) 8450 1130

由于标准和材料的变更，文中所述特性和本资料中的图象只有经过我们的业务部门确认以后，才对我们有约束。



本手册采用生态纸印刷

SC DOC 621-HEC

2005.05